VOL. 11, NO. 2, JANUARY 2016 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences


www.arpnjournals.com

DESIGN OF LOGARITHM BASED FLOATING POINT MULTIPLICATION AND DIVISION ON FPGA

N. Ramya Rani[1] V. Subbiah[2] and L. Sivakumar[1]
[1]Department of Electronics & Communication Engineering, Sri Krishna College of Engineering & Technology, Coimbatore, India
[2]Department of Electronics & Communication Engineering, PSG College of Technology, Coimbatore, India
E-Mail: ramyarani@skcet.ac.in

ABSTRACT

Logarithmic number systems (LNS) find many of its applications in the field of multimedia, digital signal processing, scientific computing and artificial neural networks due to logarithm and antilogarithm elementary functions. In this paper, logarithm based single precision floating point arithmetic units are designed based on look-up table method that computes various functions like log, antilog, rounding and exponential terms. This paper is focused in the efficient design of logarithmic floating point multiplication and division. Compared to conventional floating point arithmetic units, this work presents the design by using the same hardware for performing logarithmic operations, antilogarithm, rounding and exponential functions. Hence this work is found to be efficient in terms of area and speed compared to the design of conventional floating point arithmetic designs. Synthesis results were obtained in Xilinx SPARTAN and VIRTEX FPGA devices. Comparative results were presented for conventional floating point arithmetic units and log LUT based arithmetic designs.

Keywords: floating point multiplication, division, logarithm number system, look up table, FPGA.

INTRODUCTION

In today's scenario, digital signal processing (DSP) operations and scientific applications are considered to be some of the most challenging computations. They often need to be done in real time and require a large dynamic range. The requirements for performance and a large dynamic range lead to the use of floating point or logarithmic number systems. A wide range of scientific applications are dependent on the computation of logarithms. Logarithm based computations are used to avoid underflow by replacing multiplications via additions. Modern FPGA devices have millions of look-up tables (LUTs), registers, hardware multipliers and microprocessors. These features made the designs of floating point and logarithm based arithmetic circuits to be applicable to FPGAs. In 90's, FPGA designers implemented the design of floating point arithmetic units on FPGA devices. All these designs were concentrated in the optimization of area. Even though scientific computations prefer floating point representation compared to fixed point representation, floating point arithmetic designs has increased complexity. Hence logarithmic number systems gains advantages over floating point systems. This is because floating point multiplication and division are modified to fixed point addition and subtraction that reduces the complexity in floating point arithmetic. This work presents the design of log look up table based multiplication and division operations along with the computation of antilogarithm, rounding and exponential elementary functions.

The paper is organized as follows. Section II discusses the survey of various methods used in the design of floating point logarithmic arithmetic units. Section III provides a background on the number system representation of floating point numbers in IEEE 754 standard and logarithmic number systems. Section IV, describes the conversion of floating point value to log value and the steps to compute the antilog of a number. Section V provides the logarithm based floating point arithmetic operations and comparison with the conventional floating point arithmetic operations. Section VI details the simulation and synthesis results and comparative study of the design of conventional floating point modules with the proposed work. Finally, Section VII concludes the work and areas for future study.

LITERATURE SURVEY

G. Govindu, R. Scrofano, and V. K. Prasanna et.al proposed the design of floating point arithmetic cores. [1].Based on these cores design of arithmetic and logic units were proposed based on IEEE 754 standard. [5] G. Govindu, R. Scrofano, and V. K. Prasanna et.al proposed the area efficient architecture for the design of arithmetic expressions using the floating point cores. [2].An island-style with embedded FPU is proposed by Beauchamp *et al*., while a coarse-grained FPU was suggested by Ho *et al*. Even *et al*. suggests a multiplier for performing on either single-precision or double precision floating point numbers. [8].An optimized FPU in a hybrid FPGA was suggested by Yu *et al*. and a configurable multimode FPU for FPGAs by Chong and Parameswaran [7].Performance improvisation and optimization techniques of these suggested models are studied and analysis in terms of area and speed were provided. Anand *et al*. proposed a log lookup table (LUT)-based FPU, which utilizes a logarithmic principle to achieve good accuracy with reduced power consumption. [9]. However, this model has some serious drawbacks, which include increased delay and additional memory for the log LUT handling.[10]. The above factors affected the performance in terms of area and speed. Hence, this proposed scheme suggests an efficient model for performing floating point operations

based on logarithm look up table methods. This reduces the overall computation burden, as the process is simply a numerical transformation to the logarithmic domain.

NUMBER SYSTEMS REPRESENTATION

Floating Point Number Systems

IEEE 754 Standards

The IEEE 754 2008 Standard for Binary floating point arithmetic specifies the basic and extended floating point number formats. The basic formats of IEEE single precision and double precision floating point values are presented here.

IEEE Single Precision Format

31	30	23	22	0
Sign (1 bit)	Exponent (biased127) (8 bits)		Mantissa(23 bits)	

The floating point value for this format is given by the formula,

$$\text{Value (V)} = (-1)^{s} \times 2^{e-127} \times 1.f \quad (1)$$

The range of floating-point numbers depends on the number of bits or digits used for the representation of the mantissa and for the exponent. The range of IEEE single precision floating point number is $\pm (2\text{-}2^{-23}) \times 2^{127}$ in binary and $\sim\pm 10^{38.53}$ in decimal format.[13]

IEEE Double Precision format

63	62	52	51	0
Sign (1 bit)	Exponent (biased1023) (11bits)		Mantissa(52bits)	

The floating point value for this format is given by the formula,

$$\text{Value (V)} = (-1)^{s} \times 2^{e-1023} \times 1.f \quad (2)$$

The range of IEEE double precision floating point numbers is $\pm (2\text{-}2^{-52}) \times 2^{1023}$ in binary and $\sim\pm 10^{308.25}$ in decimal format.

Logarithmic Number Systems

In contrast to floating point numbers, in logarithmic number systems the mantissa is always 1 and the exponent has a fractional part. Let the number A has a value of

$$A = -1^{s_A} \times 2^{E_A}$$

S_A is the sign bit and E_A is a fixed point number. The sign bit represents the sign of the number and E_A represents the 2's complement fixed point number. Similarly logarithmic numbers can be represented in both very large and very small numbers.

The representation of the logarithmic numbers is as per the following format:

2 bits	1bit	K bits	1 bits
	S_A	E_A	
Flags	Sign	Integer	Fraction

Logarithmic number system based operations are very rare to identify the exceptions occurred during the computations. Hence these exceptions are identified with the representation of flag bits to specify the code for zero, +/- infinity, and NaN. [12]

The logarithmic number system has the similar range and precision of floating point number systems. For the sign bit, a=8 and the mantissa b=23bit of a single precision number, the range in LNS system is approximately $\pm 1.5\times 10^{-39}$ to 3.4×10^{38}. For the double precision floating point number, LNS system has a range of approximately $\pm 2.8\times 10^{-309}$ to 1.8×10^{308}. Thus, LNS representation covered the entire range of the corresponding IEEE 754 standard representation of floating-point numbers. [21].

CONVERSION OF FLOATING POINT VALUE TO LOGARITHMIC VALUE

Some applications require the conversion of floating point value to logarithmic value. Certain algorithms are dependent on logarithm based values while computing the floating point operations. For example, if an algorithm has a series of logarithm number system computations (multiply, divide, square root, powers) followed by a series of floating point computations (add, subtract, etc.) conversion from one form to another form will be advantageous to perform the computations efficiently. [19]

Floating Point System to Logarithm Number System (LNS)

The input floating point number based on IEEE 754 standard contains three parts namely Sign, Exponent, Mantissa and the output LNS number results in two parts namely exponent and mantissa.

Let the floating point number A can be defined as

$$A = S \times 2^{E} \times M \quad (3)$$

The conversion from floating point to LNS consists of the following steps.

i. Let the log of number A is $\log_2(A) = \log_2(2^E \times M)$
ii. By multiplication rule of log: $\log_2(A) = \log_2(2^E) + \log_2(M)$, 'E' is taken without bias.
iii. By power rule of log: $\log_2(A) = E \times \log_2(2) + \log_2(M)$
iv. As $\log_2(2) = 1$; $\log_2(A) = 1 + \log_2(M)$;

Mantissa is taken with hidden bit `1' i.e. *1.M*

v. Find $\log_2(M)$ from **Look-uptable** (for $\log_2$ of numbers between 0-1)

vi. Represent the log of number as addition of E + $\log_2(M)$ as $\log_2(A) = E + \log_2(M)$

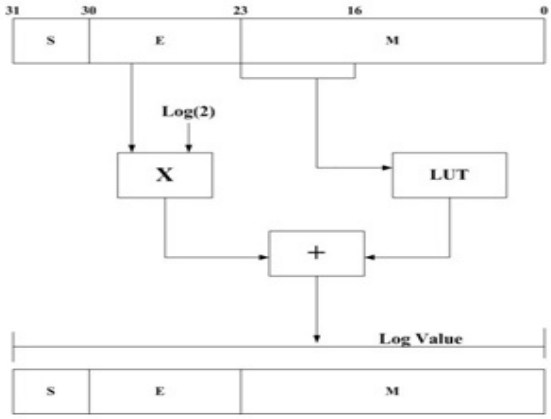

Figure-1. Floating point to logarithmic structure.

Eg: To calculate the log of floating point number
A = 32'h41CE0000 i.e. 25.75

Step 1: $\log_2(32'h41CE0000) = \log_2(2^4 \times 1.609375)$
Step2: $\log_2(32'h41CE0000) = \log_2(2^4) + \log_2(1.609375)$
Step3: $\log_2(32'h41CE0000) = 4 \times \log_2(2) + \log_2(1.609375$
Step 4: $\log_2(32'h41CE0000) = 4 + \log_2(1.609375)$
Step 5: $\log_2(1.609375) = 0.68650052718$ from Look up Table of $\log_2$
Step 6: $\log_2(32'h41CE0000) = 4 + 0.68650052718 = 4.68650052718$

The antilog is the inverse of log of a number. The anti-logarithm is defined as $\text{antilog}_2(A)=2^A$, where, A consist of exponent (E) and mantissa (M) part i.e. A=E+M

Steps to find the antilog of a log number is given below:

Step1: $\text{antilog}_2(A)=2^{As}$
Step2: $\text{antilog}_2(E+M)=2^{(E+M)}$
Step3: $\text{antilog}_2(E+M)=2^{(E)x}\,2^{(M)}$; antilogof M from look-up table (antilog of number between 1 to 2).

LOG BASED FLOATING POINT OPERATIONS

Log based Addition/Subtraction

Let the two numbers A and B be represented in LNS: The addition and subtraction of two numbers can be computed as follows in LNS systems. [17]

i) Sign of the output= Sign of A
ii) The fixed point number $E = \log_2(A \pm B)$
$= \log_2[A(1 \pm B/A)]$
$= \log_2(A) + \log_2(1 \pm B/A)$
$= E_A + f(E_B - E_A)$

Where $f(E_B - E_A)$ is defined as
$f(E_B - E_A) = \log_2(1 \pm B/A) = \log(1 \pm 2^{(E_B - E_A)})$

Thus LNS addition and subtraction computations are simpler when compared with floating point addition and subtraction arithmetic operations.

Floating point addition was difficult to implement because the significands are represented in sign-magnitude format.[3]. The difficulty arises because depending on the signs of the two operands, the addition could become a subtraction and requires one of the operands to be complemented. For addition there could be carry-out, in which the case the result would be de-normalized. For subtraction a negative result might be obtained which meant both the sign bit and the significand need to be inverted. [13]

Let the two operands be 'X' and 'Y' in IEEE 754 format (32 bit).

1. To get the sign bit (1 bit)
   - If '2' operands have similar sign bits, resultant sign bit will be the same i.e. '0' or '1'.
   - If sign bit differs, compare their exponent and mantissa fields.
     - If first operand is larger than the second, first operand sign bit will be the resultant and vice versa.
     - If both the operands have equal exponent and mantissa fields, resultant sign bit is '0'.
2. Exponent comparison (8 bits)
   - Initial resultant exponent : larger of Xe, Ye
   - Compute exponent difference
   - Make the 24th bit (hidden) explicit in the mantissa(significand)
   - If Xe >Ye, right shift the mantissa of 'Y' as per the exponent difference value computed &its exponent incremented .Fill the left most digits with zeros.
   - If Ye > Xe, right shift the mantissa of 'X' as per the exponent difference value computed & its exponent incremented .Fill the left most digits with zeros
3. Compute the sum of aligned mantissas in case of similar sign bits else subtract i.e.2's complement the mantissas in case of different sign bits.
4. Normalization of the resultant mantissas (23 bits)
   - No carry out in step '3'--- left shift to get 1.f format & exponent value decremented as per the shifts. LSB bits filled with zeroes as per the shifts.
   - If carry out results in step '3'--right shift to one place and exponent value incremented by '1' and one LSB bit gets dropped.
5. Assemble the final result into 32 bit format.

Log based Multiplication

Multiplication operation involves a simple computation in Logarithmic Number System.

The product is computed by adding the two fixed point logarithmic numbers as per the following logarithmic property:

$$\log_2(x.y) = \log_2(x) + \log_2(y) \quad (4)$$

The sign bit is computed by XORing the multiplier's and multiplicand's sign bits. The flags bits for infinities, zero, and NANs are encoded for the exceptions similar to IEEE 754 standard. Since the logarithmic

numbers are the representation of a 2's complement fixed point numbers, if overflow doesn't exists, addition is an operation similar to floating point computation. And if overflow condition occurs it will result in ±∞. Overflow condition occurs, when the addition of two numbers of same bit size resulted with the increase in bit width.

In this work, log based 32 bit floating point multiplication was computed based on look up table method. 128 values of single precision floating point numbers is called as a function and accessed as 7 bit integer. Logarithmic addition was performed for logarithmic multiplication. For the corresponding multiplication result of logarithm computation the antilogarithm of the number is computed. [16] In Floating point multiplication, the sign-magnitude format simplifies the multiplication operation because it was similar to an integer format.[5]. The only additional step required compared to addition algorithm was the calculation of correct exponent.

Let the two operands be 'X' and 'Y' in IEEE 754 format (32 bit)

1. Sign bit (1 bit)
   XOR of the two operand sign bit fields.
2. Exponent: (8 bit)
   Add the two operand's exponent and subtract the bias value'127' from their sum.
3. Mantissa computation: (24 bits)
   Multiply the 2 mantissas along with the hidden bit and the result would be in 48 bits.
4. Normalization:
   - Upper most 24 bits of the result were retained.
   - No carry in step '3'– left shift to get 1.m format and the exponent value decremented.
   - Carry in step '3'– right shift once to get 1.m format and the exponent value incremented.
5. Assemble the final result into 32 bit format.

Log based Division

The division operation of two numbers in LNS is the logarithmic subtraction of two numbers as per the following property.

$$\log_2 (x/y) = \log_2 (x) - \log_2(y) \quad (5)$$

The sign bit is computed by XORing the two operand's sign bit. The log based 32 bit floating point division was computed based on look up table method. 128 values of single precision floating point numbers is called as a function and accessed as 7 bit integer. Logarithmic subtraction was performed for logarithmic division. For the corresponding division result of logarithm computation the antilogarithm of the number is computed. [20]

In floating point division,

Non restoring method of division was used for the division of floating point numbers. Advantage of this method was restoring of the partial remainder was not required as in the case of restoring method of division. The steps are as follows:

- **Dividend Alignment**
  Compare the 2 mantissas (m1&m2) respectively of X&Y. If m1≥m2, right shift 'm1' to one place. Else shifting not required. The operand with its shifted mantissa is assigned as dividend &its exponent incremented by '1'.Other operand is assigned as divisor. This process results in normalized quotient after division.
- **Sign of Quotient:** Dividend sign XOR Divisor sign
- **Exponent of Quotient:** Subtract the divisor's exponent from the dividend's exponent & add it with the bias value.[18]
- **Non-Restore Division :**

1. Take the 2's complement of the divisor's mantissa &add it with the dividends mantissa.
2. Remainder—Left shift the above result &monitor it's MSB. If resulted in '0', 2's complement divisor value + corresponding shifted value is returned, if '1' Divisor +shifted value are returned.
3. Simultaneously Complementing the MSB while left shifting & adding, the Quotient value of division was obtained.
4. Assemble the final result into 32 bit format.

EXPERIMENTAL RESULTS

Logarithm based floating point multiplication and division was designed based on look up table method. These modules were designed in Verilog. Xilinx ISE 9.2 software was used for the synthesis of the designs and simulated using Model sim, a simulator from Model Technology (Mentor Graphics Company).Simulations results are presented here. Designs were synthesized in both Xilinx SPARTAN 6 and VIRTEX 7 FPGA devices and the device utilization report is presented here. Comparative synthesis results were obtained for the design of conventional floating point multiplication, division arithmetic units and for the proposed log LUT based arithmetic designs.

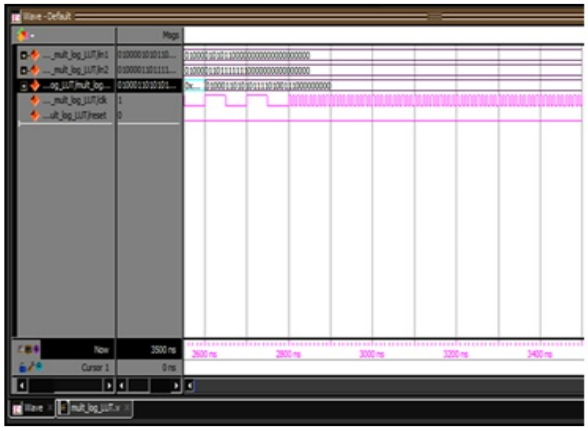

Figure-2. Simulation result of multiplication using log-LUT.

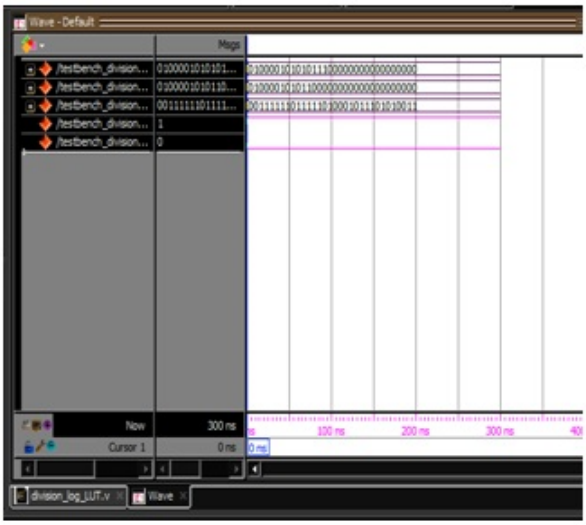

Figure-3. Simulation result of division using Log-LUT.

Table-1. Comparison between conventional and Log-LUT multiplication on VIRTEX 7 FPGA.

Device VIRTEX 7 FPGA	CONVENTIONAL FLOATING POINT MULTIPLICATION (Single Precision Multiplication only)	LOG-LUT MULTIPLICATION (Single Precision) (Antilog,Round& Exponential)
Area (slices)	102 out of 385600	318 out of 385600
Clock Speed (MHZ)	183.093MHZ	97.066MHz

Table-2. Comparison between conventional and Log-LUT division on VIRTEX 6 FPGA.

Device VIRTEX 6 FPGA	CONVENTIONAL FLOATING POINT DIVISION (Single Precision Division only)	LOG-LUT DIVISION (Single Precision) (Antilog,Round& Exponential)
Area (slices)	1346 out of 46560	11688 out of 46560
Clock Speed (MHZ)	22.240MHz	7.501MHz

Table-1 and 2 showed that, the log lookup table based multiplication and division operations that included antilogarithm, rounding and exponential functions required area comparatively more than that of the implementation of only single precision floating point multiplication and division operations. Also the speed of execution was high for conventional floating point operations compared to log LUT designs. Even though from the tables it was seen directly that proposed log LUT design occupied more area and lesser speed of execution but with the inclusion of other elementary functions the proposed designs were proved to be advantageous than the conventional designs.

Table-3. Comparison between Spartan6 and Virtex6 FPGA for multiplication.

Device	SPARTAN 6 FPGA		VIRTEX 6 FPGA	
Precision	Latency (ns)	Clock Speed (MHZ)	Latency (ns)	Clock Speed (MHZ)
Conventional FP Multiplication	2.113ns	48.185 MHz	1.56ns	138.794 MHz
LOG-LUT based Multiplication	340.187 ns	2.937 MHz	146.377 ns	6.898 MHz

VIRTEX 6 FPGA device has higher gate density, more number of 18×18 multipliers, embedded Harvard architecture block compared with SPARTAN 6 FPGA devices. This advantage of the VIRTEX 6 FPGA processors allowed for the faster implementation of the conventional designs, log LUT designs and also provided less latency compared to SPARTAN 6 devices. These parameters were proved in Table-3. The Register Transfer Level Schematic (RTL) was given below for log LUT multiplication designs. Similarly the schematic can be obtained for log LUT division designs [24].

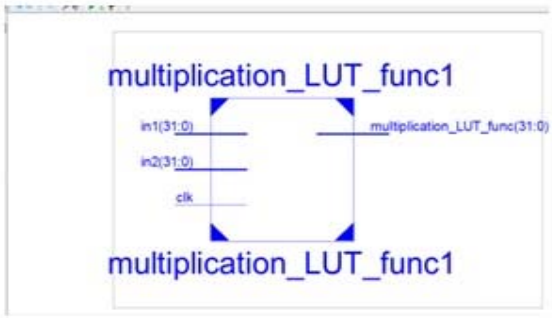

Figure-4. RTL schematic block-Log LUT multiplication.

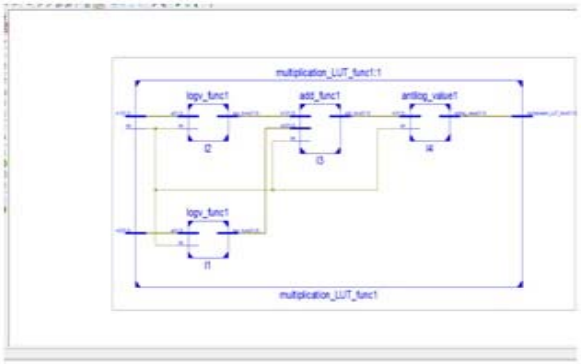

Figure-5. RTL sub block schematic –Log LUT multiplication.

CONCLUSION AND FUTURE WORK

This paper presented the design of log LUT floating point multiplication and division unit. The module was designed based on direct look up table method. From the experimental results it was concluded that, the log LUT based floating point multiplication and division designs with the inclusion of elementary functions consumed comparatively less area and high speed of operations than the conventional floating point arithmetic designs.

In future the work can be implemented in the field of image processing. Also the work can be extended to design and implement the embedded logarithmic arithmetic units on FPGA.

REFERENCES

[1] G. Govindu, R. Scrofano, and V. K. Prasanna, "A library of parameterizable floating-point cores for FPGAs and their application to scientific computing," in Proc. Int. Conf. Eng. Reconfigurable Syst. Algorithms, 2005, pp. 137–148.

[2] R. Scrofano, L. Zhuo and V.K. Prasanna, "Area-Efficient Arithmetic Expression Evaluation Using Deeply Pipelined Floating-Point Cores," IEEE Transactions on VLSI systems,vol.16,no.2,pp.167-176,2008.

[3] Zhaolin Li, Xinyue Zhang, GongqiongLi, Runde Zhou, "Design of A Fully Pipelined single-precision Floating-Point Unit", IEEE,2007,pg 60-63.

[4] Mingjie Lin, Shaoyi Cheng, John Wawrzynek, "Cascading Deep Pipelines to Achieve High Throughput in Numerical Reduction Operations", IEEE 2010, Pg.103-108.

[5] IEEE Standard for Floating-Point Arithmetic, 2008.

[6] G Even, SM Mueller, P-M Seidel, A dual precision IEEE floating-point multiplier. Integration. VLSI J. 29(2), 167–180 (2000).

[7] CW Yu, AM Smith, W Luk, PHW Leong, SJE Wilton, Optimizing floating point units in Hybrid FPGAs. IEEE Trans. Very Large Scale Integer (VLSI) Syst. 20(7), 45–65 (2012).

[8] YJ Chong, S Parameswaran, Configurable multimode embedded floating point units for FPGAs. IEEE Trans. Very Large Scale Integer (VLSI) Syst.19 (11), 2033–2044 (2011).

[9] TH Anand, D Vaithiyanathan, R Seshasayanan, "Optimized architecture for floating point computation unit", in Int. conf. on emerging trends in VLSI, embedded sys, nano elec. and tele. sys (Thiruvannamalai, India, 2013), pp. 1–5.

[10] S Paul, N Jayakumar, SP Khatri, A fast hardware approach for approximate, efficient logarithm and antilogarithm computations. IEEE Trans. Very Large Scale Integration (VLSI) Syst. 17(2), 269–277 (2009).

[11] Haohuan Fu, Member, IEEE, Oskar Mencer, Member, IEEE, and Wayne Luk, Fellow, IEEE, "FPGA Designs with Optimized Logarithmic Arithmetic", IEEE Transactions On Computers, VOL. 59, NO. 7, JULY 2010.

[12] Nikolaos Alachiotis, Alexandros Stamatakis, "Efficient Floating-Point Logarithm Unit for Fpgas" in IEEE, 2010.

[13] MJ Beauchamp, S Hauck, KD Underwood, KS Hemmert, "Architectural modification to enhance the floating-point performance of FPGAs", IEEE Trans. Very Large. Scale Integer (VLSI) Syst. 16(2), 177–187 (2008).

[14] CH Ho, CW Yu, PHW Leong, W Luk, SJE Wilton, "Floating-point FPGA: architecture and modeling". IEEE Trans. Very Large Scale Integer (VLSI) Syst.17 (12), 1709–1718 (2009).

[15]Ramin Tajallipout,Mf.Asraful Islam and Khan A.Wahid, "Fast Algorithm of a 64 bit Decimal Logarithmic Converter", Journal of Computers,vol.5,No.12,December 2010.

[16] Mark G.Arnold and Sylvain Collange, "A Real/Complex Logarithmic Number System ALU",IEEE Transactions on Computers, Vol.60,No.2,February 2011.

[17] H. Fu, O. Mencer, and W. Luk, "Optimizing Logarithmic Arithmetic on FPGAs," Proc. IEEE Int'l Symp. Field-Programmable Custom Computing Machines (FCCM), pp. 163-172, 2007.

[18] M. Haselman, M. Beauchamp, K. Underwood, and K. Hemmert, "A Comparison of Floating Point and Logarithmic Number Systems for FPGAs," Proc. IEEE Int'l Symp. Field-Programmable Custom Computing Machines (FCCM), pp. 181-190, 2005.

[19]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.

[20] B. Lee and N. Burgess, "A Parallel Look-Up Logarithmic Number System Addition/Subtraction Scheme for FPGA," Proc. Int'l Conf. Field-Programmable Technology (FPT), pp. 76-83, 2003.

[21] Mark G. Arnold, Member, IEEE, and Sylvain Collange, "A Real/Complex Logarithmic Number

System ALU", IEEE Transactions On Computers, Vol. 60, No. 2, February 2011.

[22]M.G. Arnold and S. Collange, "A Dual-Purpose Real/Complex Logarithmic Number System ALU," Proc. 19th IEEE Symp. Computer Arithmetic, pp. 15-24, June 2009.

[23]J. Muller, Elementary Functions: Algorithms and Implementation. Springer, 2006.

[24]Virtex-6 Family Overview, Xilinx, Inc., http://www.xilinx.com, 2007.