International Journal of Engineering & Technology, 7 (3.20) (2018) 616-619

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

FPGA implementation of Leading One Detector using Genetic Algorithm

Kishore Kumar ATA [1*], Dr.Seshasayanan R [2]

[1]*Associate Professor, ECE Dept, KMM Institute of Technology & science, Thirupathi, Andhra Pradesh, India, Research Scholar, Sathyabama University, Chennai, India.*
[2]*Associate Professor, Dept of ECE, Anna University, Chennai, India*
**Corresponding Author Email:kishorettt@gmail.com ,se_sha_sa@yahoo.com*

Abstract
Logarithmic conversion is a significant portion of numerous digital signals processing system, particularly in the fields of instruments design. Twelve bits of fractional accuracy demands lesser memory usage and minimum arithmetic components. The logarithmic transformation presented in this paper is able to support the logarithmic conversion of data with the number of bits up to sixteen. The proposed work circles around Look up Table (LUT) based approach and follow a decimal linear estimation step. The implementation results shows that the proposed architecture will operate at 43.7 MHz in FPGA fabric and at 101.9 MHz in TMSC 0.18-um technology.

Keywords: *LOD, Genetic Algorithm, FPGA, LNS.*

1. Introduction

A wide variety of logarithmic calculation is applied in the scientific applications. Multimedia codes e.g., Gaussian Mixture Models are to be estimated or Bioinformatics codes for evolutionary rebuilding under the Maximum Likelihood model [1, 2] demand calculating log probability scores of evolutionary trees. Multiplications via additions are normally used in logarithm to take care of underflow problems. Compute-intensive applications which rely on the logarithmic or skin segmentation algorithms like Phylogenetic likelihood function [3 – 5] or real-time image processing applications with real-time constraints. FPGA specific implementations house any type of applications to harness features such as improved speed, hardware prototype design or meeting real time constraints. Phylogenetic co-processor for RAxML based on Maximum Likelihood (ML) model is carried out within the design and framework of research [6,11]. Logarithmic function implementation based on FPGA is available in new generation FPGAs [7]. The function FloPoCo [8] is used to generate the above-mentioned logarithm implementation.[17,18]

The above implementation delivers extraordinary arithmetical precision, it might not be required always due to less clock frequency. So, a remarkably quicker with slightly less precise logarithm implementation is often desirable depending upon the applications. The proposed work discusses a reconfigurable architecture for floating point logarithm approximation operating at higher frequency compared with existing FPGA implementation[15]. Subsequently the logarithm is an important function and the proposed unit will find a place for extensive range of applications even beyond the scope of our research. Naturally most common computation challenging algorithms are Digital signal processing (DSP). It requires a large dynamic range of numbers and it should be done in real time environment[16].
Logarithmic or floating point representation is imperative for improved performance to handle large dynamic range. Mid 90's saw the advent of usage of FPGAs for mapping floating point arithmetic units [Fagin94, Louca96, Belonovic02, Underwood [04]. It is a trade-off between the hardware complexity and the precision attained with these implementations. Floating point representations achieves dynamic range but fails as it is less precise together with hardware complexity. Fixed point scores better against the floating point representations in the above counts. Logarithmic number system (LNS) is seen to achieve precision and a comparable range but fails when it comes to hardware complexity again. The multiplication and division are often simplified to addition and subtraction in LNS but in few of the cases, floating point number representations have been standard in LNS.

2. Backgrounds

2.1 Earlier Work

Study on floating point arithmetic units in FPGA catapulted with the release of IEEE compliant floating point units [Fagin94]. IEEE compliant realizations can be glanced in [Fagin94, Louca96, Underwood04]. There are also implementations using variable word sized due to flexibility of FPGAs. This provides the platform for application specific optimizations [Belanovic02, Ho02, Liang03, Liang03]. Incremental improvement in throughput and area is achieved by optimizing the addition, subtraction, multiplication, division and square root operations separately [Loucas96, Li97, Wang03]. Lastly, realizations of floating point arithmetic units in FPGA proved that in addition to introducing arithamtic operations, several useful applications could well be carried out with FPGAs [Walter98, Leinhart02]. Off late, research on logarithmic number system resulted in employing the same in the place of floating point units.

Logarithmic microprocessor has been proposed [Colmen00]. The complexity associated with the employability of LNS has prompt-

Copyright © 2018 Authors. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

ed researchers to write algorithms for LNS addition/subtraction [Lewis93, Colmen00, Lee03] and conversion from floating point to LNS [Wan99, Wan99, Abed03]. Comparison of floating point to LNS is reported in [Colmen99, Matousek02, Detrey03]. Colmen et al. discusses the accuracy of LNS addition with that of floating point addition by comparing the delay in ASIC environment. Delay comparison for a 20-bit and 32-bit word size in LNS and floating point representation remain bleak in the context of usage of both [Matousek et al]. Limitations of LNS library has led to the implementations of floating point units for 20 bits and lesser bit widths.

Logarithmic numbers is a special case of floating point numbers where the exponent has a fractional part and the mantissa is permanently 1 [Koren02]. The value of the number A is

$$A = -1^{s} \times 2^{E}$$

where S represents sign bit and the fixed point number is E. The whole number sign is represented in the sign bit. EA represents fixed point number in 2's complement and represent negative numbers by less than 1.0 value. Thus, LNS number system proved good in representing small or big numbers.

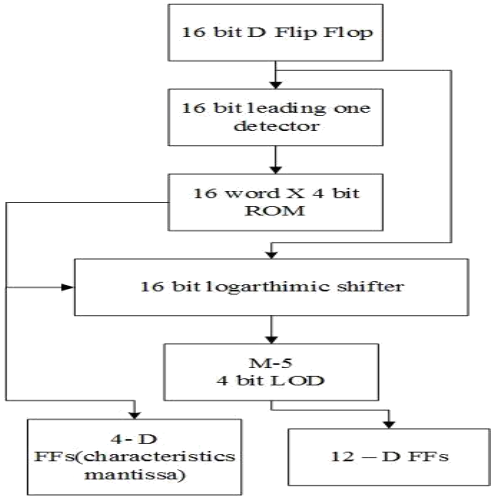

Figure 1: Logarithmic number format

There is no specific choice to represent special values such as exceptions, and zero which is not represented in LNS, so in this paper we decided to adopt similar manner as Detrey et al [Detrey03] to represent NaN, +/- infinity and zero using flag bits.

2.2 Multiplication

Addition of two fixed point logarithmic numbers yielded their product in LNS [Koren02]. Here the product of two numbers is obtained by adding the two fixed point logarithmic numbers which is shown in equation 1.

$$\log_2(x,y) = \log_2(x) + \log_2(y) \quad (1)$$

XORing the sign bit of the multiplicand and the multiplier gives the product sign. The flags for NANs, zero and infinities are encoded for exceptions as per the IEEE 754 standard. The logarithmic numbers are represented in 2's complement fixed point numbers and the addition is exact if overflow flag is not set, otherwise result in $\pm\infty$. The overflow flag is set due to addition of two numbers becoming too large compared to word width length.

The two mantissas are multiplied and exponent terms are added in the case of floating point multiplication [Koren02]. Using the following property the exponent terms are added which is shown in equation 2:

$$2^{x} \times 2^{y} = 2^{x+y} \quad (2)$$

The exponents are having bias component and one bias is subtracted from the addition result. Integers are used in the exponents and there is a probability that the exponent's addition will produce an integer. This sets the overflow flag as it is huge to keep it in exponent field and infinity exceptions is set. The range for two mantissas are [1,2], the resultant product will have the range between [1,4) and for the mantissas renormalization one right shift is possible. Increment is made in the exponent for a right shift of the mantissa and another probability of overflow detection.

2.3 Division

Using the following equation 3 division in LNS becomes subtraction based on logarithmic property [Koren02]:

$$\log_2 (x/y) = \log_2 (x) - \log_2 (y) \quad (3)$$

Multiplication sign and the division sign, which are one and the same, is computed by XORing the sign bits of the two operands. Difference causes the probability of the underflow to equal zero due to subtraction. The result of the subtraction is larger than the word width representation the underflow occurs. Floating point division is obtained by subtracting the exponent terms [Koren02] and dividing divisor's mantissas by dividend's mantissas. The mantissas range is [1,2], quotients range is [0.5,2] and mantissa requires one left shift for renormalization. One left shift means the exponent decrement and underflow detection.

2.4 Logarithmic Converters

The 16 bit logarithmic converter is proposed and it is shown in Figure 2. The result of the logarithms of a 16 bit binary number by this implementation is attained by one clock cycle. The logarithmic converter implementation using hardware has two stages. The stage one contained 16-bit shift register and LZD. The first stage results in 4- digit MSB of input; the second stage consists of the error rectification logic which completes the linear approximation of decimal logarithmic with a size of 512 ×16 coefficients ROM.

Table 1 Approximate two digit decimal conversion for Logarithmic conversion

N Decimal	N Binary	Log N Exact	Log N Approx	Log N Binary Approx
1	00001	0	0	000.0000
2	00010	1	1	001.0000
3	00011	1.5849625	1.5	001.1000
4	00100	2	2	010.0000
5	00101	2.3219280	2.25	010.0100
6	00110	2.5849625	2.5	010.1000
7	00111	2.8073549	2.75	010.1100
8	01000	3	3	011.0000
9	01001	3.1699250	3.125	011.0010
10	01010	3.3219280	3.25	011.0100
11	01011	3.4594316	3.375	011.0110
12	01100	3.5849625	3.5	011.1000
13	01101	3.7004397	3.625	011.1010
14	01110	3.8073549	3.75	011.1100
15	01111	3.9068905	3.875	011.1110
16	10000	4	4	100.0000
17	10001	4.087463	4.0625	100.0001

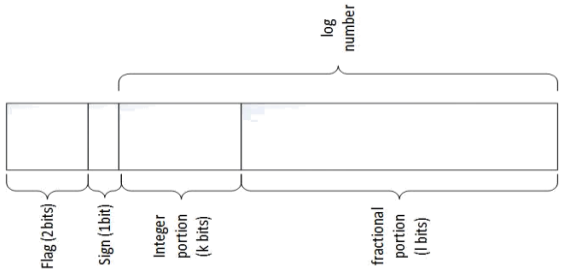

Figure 2: Flow chart for the 2 digit decimal

2.5 Fitness Function

The fitness function for each chromosome is derived as fallows. The out puts are calculated for the given set of possible inputs. Then these outputs are compared with the actual outputs and the degree of similarity is obtained. This constitutes the fitness value for a given individual chromosomes. In agreement with the statement "survival of the fittest", the chromosomes have better fitness is taken to the next generation iteratively till the required fitness is obtained. Fitness function is used to compute individual fitness in a population and depicted as a table below for 4-bit LOD is shown below.

Table 2 Fitness Table representation of 4-bit LOD

Input				Output				Zero Flag
X3	**X2**	**X1**	**X0**	**Y3**	**Y2**	**Y1**	**Y0**	**V**
0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	1	0
0	0	1	X	0	0	1	0	0
0	1	X	X	0	1	0	0	0
1	X	X	X	1	0	0	0	0

2.6 Population

Out off the initially randomly selected 100 chromosomes, the best 50 chromosomes are selected based on the fitness function to represent the initial population. By this way a random initial population is used as a building block for a functional combinational circuit in full viz a 4-bit evolved LOD.

2.7 Evolved LODS

In the logarithmic representation of numbers, the integer and the fractional pars are determined as fallows. The leading one bit position determines the integer part. The LOD output is used to shift the input, their by determining the fractional part. A circuit utilizing less hardware consumes less power and operating with high speed to detect the leading one bit position is therefore the need of the hour. This LOD outputs a logic one given input leading one and zero otherwise. The building block for the LOD discussed by

Oklobdzija(1992) is a multiplexor. This helps in lessening the delay. The evolved 4-bit LOD is used to construct higher order LODs and it was discussed by Abed & Siferd (2006). To reduce delay we have replace 2:1 multiplexer with logic gates. In the proposed method lower order LOD circuit is evolved using the 2-input logic gates derived from basic logic gates. In figure3, the best know evolved 4-bit LOD is shown.

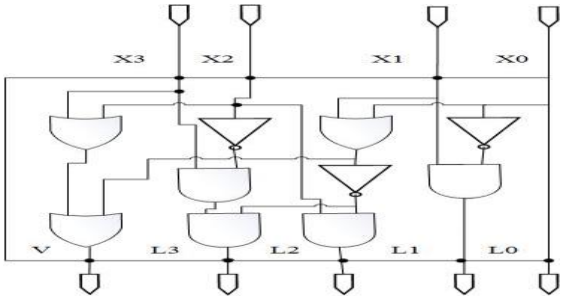

Figure:3: Evolved four bit LOD

In the proposed method, the lower order LOD circuit is evolved using the 2-input logic gates derived from the basic logic gates. Higher order LODs are considered with evolved 4 bit LOD as illustrated by Abed &Siferd (2006). In figure3, the best know evolved 4-bit LOD is shown.

2.8 Mutation in 4-bit LOD

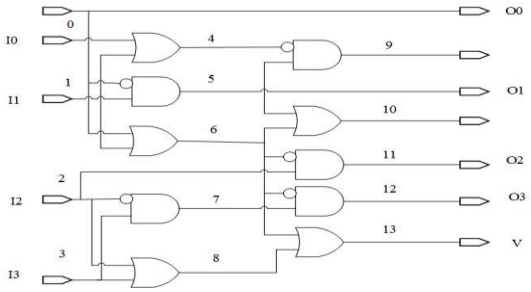

Figure 4: An example of point-mutation operator in LOD

In figure 4 the first output O0 is connected to the node 0 which corresponds to the input terminal I0.The second output O1 is connected to node 5. Node 5 is the output of bubbled AND gate (0) whose input finds node 0 and node 1.The third output O2 is connected to node 11. Node 11 is the output bubbled and gate (0) whose input is connected to node 6 and node 2. Node 2 is the input I2 and Node 6 is the output of an OR gate (1) whose inputs are node 0 and 1 which corresponds to I0 and I1 respectively. Similarly the fourth output O3 has three node 6,7 and 12 which are the output of three logic gates as shown in the figure. Output node in a chromosome changes the overall functionality of the circuit.

2.9 8-bit LOD

Two 4-bit LOD's along with eight 2-input AND gates are used to construct an 8-bit LOD. The enable signal to the AND gates in the output stage is provided by the most significant 4-bit LOD zero flag. Hence the output stage multiplexer is replaced by a set of eight 2-input AND gates. Their by reducing the overall latency of the 8-bit LOD.

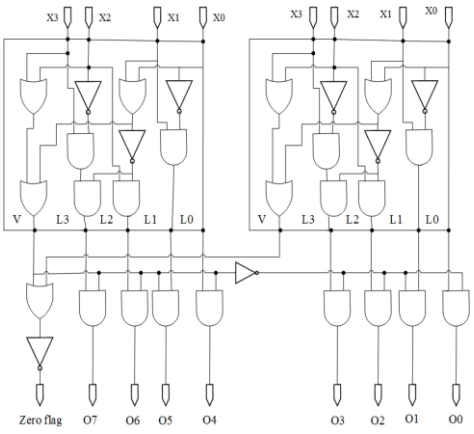

Figure 5: 8-bit LOD constructed using the evolved 4-bit LOD

3. Comparison of Synthesis Results of Lod

The different parameters of the LOD circuit built using 4-bit and 8-bit LOD are cell area and power Delay Product. The tools used for the synthesis and analysis of cell area , delay and power consumed by the LOD architectures is by Candence® RTLCompiler® with TSMC 180nm library.

The LOD architecture of different sizes are shown in the below table 5. Evolved architecture depict bettered improvement in power, delay product along with area.

Table 5: Comparison Delay and power of various LOD circuits

Size of the circuit	Power (nW)			
	Oklobdzija (1992)	Abed et al (2006)	Yemiscioglu et al	Proposed method (LOD)
16-bit	2363.02	2154.33	2067.06	1706.15
32-bit	4255.32	3786.71	3589.62	2795.28
64-bit	7956.52	7567.19	7168.74	5468.40

Size of the circuit	Delay (ps)			
	Oklobdzija (1992)	Abed et al (2006)	Yemiscioglu et al	Proposed method (LOD)
16-bit	821	854	843	831
32-bit	956	1021	994	978
64-bit	1254	1467	1402	1303

3.1 Synthesis Results of Logarithmic Converter

Device utilization summary:

Selected Device : 2s15cs144-6

Number of Slices:	71 out of	192	36%
Number of Slice Flip Flops:	12 out of	384	3%
Number of 4 input LUTs:	126 out of	384	32%
Number of bonded IOBs:	49 out of	90	54%

Timing Summary:

Speed Grade: -6

Minimum period: No path found
Minimum input arrival time before clock: 25.056ns

Maximum output required time after clock: 10.695ns
Maximum combinational path delay: 22.931ns

4. Conclusion

Various sizes of LOD are designed using automatic and evolutionary hybrid approaches are discussed. The design of higher order LODs from the evolved lower order LODs are highlighted. The performance benefits in terms of cell area and power are seen in gate level evolution of LOD circuits and shows a maximum improvement of 48.8% in the power-delay product and an improvement of 45.3% in the cell area. Cadence® RTL Compiler® with TSMC 180nm library is used in the synthesis and performance comparison of the synthesized results. The convergence of evolutionary algorithms takes more time as the complexity of the circuit increases and the evolutionary algorithm may settle for a local maxima or local minima. The proposed shuffling mechanism introduced in this GA addresses this problem and makes sure that the solution converges to a global minimum. The GA can be implemented in a reconfigurable system where the required circuits are evolved in the runtime and also fulfilling the user demands. A comprehensive analysis on the convergence of the proposed GA and its hardware implementation are needed to make it suitable for implementing this GA in a real-time system.

References

[1] R. E. Siferd and K. H. Abed , "CMOS VLSI implementation of a low-power logarithmic converter," in *IEEE Transactions on Computers*, vol. 52, no. 11, pp. 1421-1433, Nov. 2003

[2] Dongdong Chen, Younhee Choi, Li Chen, D. Teng, Khan Wahid and Seok-Bum Ko, "A novel decimal-to-decimal logarithmic converter," *2008 IEEE International Symposium on Circuits and Systems*, Seattle, WA, 2008, pp. 688-691.

[3] M. Combet, H. L. Verbeek and Van Zonneveld, "Computation of the Base Two Logarithm of Binary Numbers," in *IEEE Transactions on Electronic Computers*, vol. EC-14, no. 6, pp. 863-867, Dec. 1965

[4] M. F. Cowlishaw, "Decimal floating-point: algorism for computers," *Proceedings 2003 16th IEEE Symposium on Computer Arithmetic*, 2003, pp. 104-111

[5] J.-P. Deschamps, G.J.A. Bioul, G.D. Sutter, Synthesis of Arithmetic Circuits: FPGA, ASIC and Embedded Systems (Wiley, Hoboken, 2006)

[6] M. A. Erle and M. J. Schulte, "Decimal multiplication via carry-save addition," *Proceedings IEEE International Conference on Application-Specific Systems, Architectures, and Processors. ASAP 2003*, 2003, pp. 348-358.

[7] E. L. Hall, D. D. Lynch and S. J. Dwyer, "Generation of Products and Quotients Using Approximate Binary Logarithms for Digital Filtering Applications," in *IEEE Transactions on Computers*, vol. C-19, no. 2, pp. 97-105, Feb. 1970.

[8] IEEE, Inc., IEEE 754-2008 Standard for Floating-point Arithmetic (2008)

[9] T. Lang and A. Nannarelli, "A Radix-10 Combinational Multiplier," *2006 Fortieth Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, 2006, pp. 313-317.

[10] J. N. Mitchell, "Computer Multiplication and Division Using Binary Logarithms," in *IRE Transactions on Electronic Computers*, vol. EC-11, no. 4, pp. 512-517, Aug. 1962.

[11] Nagarajan, G., and R. I. Minu. "Multimodal fuzzy ontology creation and knowledge information retrieval." Proceedings of the International Conference on Soft Computing Systems. Springer, New Delhi, 2016.

[12] J.M. Muller, Elementary Functions, Algorithms and Implementation (Birkhauser, Boston, 2005)

[13] S. L. SanGregory, C. Brothers, D. Gallagher and R. Siferd, "A fast, low-power logarithm approximation with CMOS VLSI implementation," *42nd Midwest Symposium on Circuits and Systems (Cat. No.99CH36356)*, Las Cruces, NM, 1999, pp. 388-391 vol. 1.

[14] T. Sasao, S. Nagayama and J. T. Butler, "Numerical Function Generators Using LUT Cascades," in *IEEE Transactions on Computers*, vol. 56, no. 6, pp. 826-838, June 2007.

[15] MuhammedShafi. P,Selvakumar.S*, Mohamed Shakeel.P, "An Efficient Optimal Fuzzy C Means (OFCM) Algorithm with Particle Swarm Optimization (PSO) To Analyze and Predict Crime Data", Journal of Advanced Research in Dynamic and Control Systems, Issue: 06,2018, Pages: 699-707

[16] Selvakumar, S & Inbarani, Hannah & Mohamed Shakeel, P. (2016). A hybrid personalized tag recommendations for social E-Learning system. 9. 1187-1199.

[17] Nagarajan, G., et al. "Hybrid Genetic algorithm for medical image feature extraction and selection." Procedia Computer Science 85 (2016): 455-462.

[18] Elangomenan, P., and G. Nagarajan. "Fuzzy-Based Multiloop Interleaved PFC Converter with High-Voltage Conversion System." Proceedings of the International Conference on Soft Computing Systems. Springer, New Delhi, 2016.