A Fixed-Point Natural Logarithm Approximation Hardware Design Using Taylor Series

Miguel R. Weirich†, Guilherme Paim*, Eduardo A. C. da Costa†, Sergio Bampi*

*Graduate Program on Microelectronics (PGMicro) - Federal University of Rio Grande do Sul (UFRGS)
Porto Alegre - Brazil
†Graduate Program on Electronic Engineering and Computing - Catholic University of Pelotas (UCPel)
Pelotas - Brazil

***Abstract*—The logarithm function is employed in several areas of knowledge because its curve approximates various physical and chemical phenomena. In this work, we propose an operator which can approximate the logarithm function at the base *e*. Such an operator is implemented using Taylor Series approximation with five terms. We also offer an algorithm responsible for allowing input values different from the ones that are included in the convergence region of the Taylor Series. Through co-simulations using both the Matlab® and ModelSim® simulators, it was possible to determine that the approximation error remains less than 0.6% for the entire input range. The input range used in this paper is in integer format with a one-byte length, and the system output has 19 bits with 4.15 form.**

***Keywords*—Natural Logarithm; Taylor Series; Hardware.**

I. Introduction

John Napier firstly proposed the concept of logarithm at the end of the 17th century [1]. The original idea emerged to reduce the complexity of operations, such as divisions replaced by subtraction, multiplication replaced by addition, exponential replaced by multiplication, and root replaced by division [1]. Along the time, several studies have shown that logarithmic curves describe several natural phenomena in the most varied areas of knowledge, such as in geography to calculate the rate of population growth, or in chemistry, where a logarithm function calculates the radioactive decay curve. In image processing, more precisely in tone mapping operators, the logarithm operator is used to calculate the geometric mean of the image luminance [2], and this is the main reason that led us to develop this operator in this work.

Implementations of logarithmic functions are performed using numerical approximation methods. The most commonly used approaches include Taylor Series, CORDIC, Improved Parabolic Synthesis and Newton-Raphson [3]. This paper presents the development of a logarithmic operator at the base *e* approach by Taylor Series with an unsigned integer eight- bit input range, which provides values between 0 and 255. The operator output has 19 fixed-point bits in the 4.15 format.

The main contributions of this work are: (a) the implementation of a Taylor Series to addresses a base *e* logarithm operator, and (b) the extension of the input range of the operator for entries with range $0 < x \leq 255$. Note that initially, the operator receives values in the field of $0.5 \leq x \leq 1$, which is the Taylor Series convergence constraint.

II. Background

A. Logarithm Overview

In mathematics, the logarithm is the inverse function to exponential. It means that the logarithm of a given number x is the exponent to another fixed number, where the base b, must be raised, to produce that number x. In the simplest case, the logarithm counts repeated multiplication of the same factor [4]. The equation 1 shows the relation between the logarithmic and exponential functions.

$$log_b(x) = y <=> b^y = x \tag{1}$$

The curve that describes the logarithm function changes according to the base used. Bases greater than one produce ascending curves, and fractional bases, in the interval $0 < x < 1$, produce descending curves. Figure 1 shows upward and downward curves of the logarithm function [3].

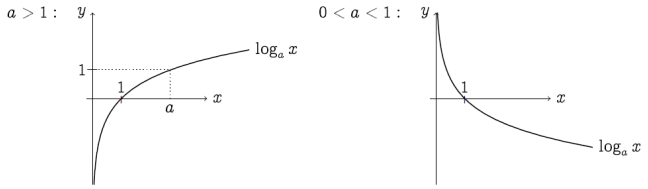

Fig. 1: Log curves (a) base a > 1; (b) base 0 < b < 1.

B. Taylor Series Overview

Taylor Series is a representation of a function as an infinite sum of terms which are calculated from the values of the function's derivates at a single point [4]. The Taylor series of a function $f(x)$ is a power series which can be expressed as 2.

$$f(x) = \sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x-a)^n \tag{2}$$

Where $n!$ denotes the factorial of n and $f^n(a)$ denotes the nth derivate of f evaluated at the point a. The derivate of order zero of f is defined to be f itself and $(x-a)^0$ and 0! are both defined to be 1 [4].

According to the literature, as in [3], 3 gives the Taylor Series that approximates the logarithm function, and the number of coefficients used in the series will define how close the result obtained will be to the actual value. In other

978-1-5386-7681-3/18/$31.00 ©2018 IEEE

words, the higher the number of factors, the less error will be achieved. In this article we use a Taylor Series with five terms because it has a point of balance between number of operations and error presented by the circuit.

$$f(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{n+1} x^{(n+1)}, |x| < 1 \quad (3)$$

C. Related Work

By reviewing works from the literature, we can find several ways of approaching the logarithm function. As an example, the work in [5] uses Mitchell approximation to propose a low cost implementation and [6] improves it. Another method widely used to approximate functions is the CORDIC (COordinate Rotation Digital Computer) algorithm. The works in [7] and [8] used this algorithm in their works. Based on the Improved Parabolic Synthesis, [9] obtained results that surpassed consolidated techniques like the CORDIC and the Taylor Series. Using Leading One Detector and Encoder (LODE) algorithm, [10] propose ASIC implementation for logarithm approximation. The last paper presented by [11] proposes an improved decimal floating-point logarithmic converter based on selection by rounding. Table I summarizes the main differences between the proposed work and the solutions from the literature.

TABLE I: Summary of Hardwired Natural Logarithm RW.

Related Work	A	B	C	D	E	F	G	H
[5]		✓	✓		✓		✓	
[6]	✓		✓		✓		✓	
[7]		✓	✓		✓		✓	
[8]		✓	✓			✓		✓
[9]	✓			✓			✓	
[10]	✓			✓			✓	
[11]	✓			✓				✓
This work		✓	✓	✓		✓		

(A) Float-Point implementation, (B) Fixed-Point implementation
(C) FPGA-based solution, (D) ASIC-based solution
(E) Using ROM or LUT, (F) $\log_e$,(G)log_2,(H)log_{10}.

III. Proposed Natural Logarithm Based on Taylor Series Approximation

The proposed operator was developed based on equation 3. As seen previously, the convergence region of the Taylor Series is $0.5 \leq x \leq 1$. As the proposal is that the input of the operator can receive values between 0 and 255, thus a second algorithm is necessary to adapt the data so that it does not leave the convergence region of the series. In the following subsections will be presented the algorithms used in the development of the operator.

A. Input signal adjustment

Two mathematical properties of logarithms must be used to adjusts the operator input. The first is ownership of the product. The equation 4 is given for this property, where x is the value between 0.5 and 1, and c is a value that multiplied by x results in the input value of the operator [4].

$$ln(cx) = ln(c) + ln(x) \quad (4)$$

With this operation, it is already possible to calculate the portion $ln(x)$ of the equation by applying it directly to the Taylor Series. To obtain the value of x, in the desired range, we need to divide the input by the value corresponding to the upper bit to the high level of the signal. For example, the decimal number 60, that in decimal corresponds to the binary number "00111100", will be divided by the binary number 01000000, which corresponds to the decimal 64. The result of this division will have the value within the convergence region of the Taylor Series.

The $ln(c)$ portion still needs to be worked on, and for this purpose, another property is used. Since implementations are performed in binary code, all signal values are expressed in the form s^k. Then using the power property of logarithms, according to equation 5, thus the value c is equal to the multiplication of k by a constant $ln(2)$.

$$ln(c) = ln(s^k) = k * ln(2) \quad (5)$$

The operator response is related to equation 6. Analyzing the application of these properties there is a scenario in which a complex operator is calculated using a divider combined with sums and multiplications.

$$ln(cx) = k * ln(2) + ln(x) \quad (6)$$

The last step of calculating $ln(cx)$ is to find the value of k who is the number corresponding to the most significant bit of the input signal with the value equal to 1. As an example, the number 7 in decimal corresponds to the number 00000111 in binary, and the MSB with the value equal to 1 is in the position of the bit 2. Thus, the value of k will be equal to 2.

B. Hardware Implementation

Figure 2 shows the block diagram of the operator. The first step is to find the value of k and the denominator to produce a division result within the convergence region of the Taylor Series. For this purpose, a series of conditionals is realized, which, according to the input, determine the outputs k and x. The divisor used in the implementation is a low power operator based on a Newton-Raphson approximation. The word length of the divider output has 16 bits in 1x15 format.

The multiplier that operates $k * ln(2)$ is a multiplication of a variable by a constant since the portion $ln(2)$ does not change its value over the time. The multiplications by constants are easily realized in hardware implementation by sums and shifts, with a low impact on system performance. Besides, this type of architecture is known to be a low-power way to realize multiplications by constants.

The calculation of $ln(x)$, follows the solution of the Taylor series. The implementation was done using five terms of the series. Then, the equation 7 establishes the series that will be implemented.

$$ln(x) = 1 + \frac{x}{1} - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} \quad (7)$$

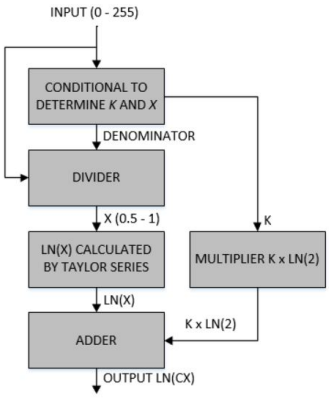

Fig. 2: Block diagram of the logarithm operator.

The implementation of the $ln(x)$ parcel was performed using parallel architecture, aiming at a faster calculation speed. The values of the coefficients were shifted to operate at fixed-point, resulting in the equation 8. The number 1 in the equation 8 is necessary to prevent the occurrence of singularity if the input value is zero. In these cases, the output of the polynomial will be zero. The output of the polynomial must receive the reverse bit shift to the initial offset so that the results match the reality. The figure 3 shows the block diagram of the Taylor Series. Analyzing the operator, the only multiplications that are not of the variable per constant type are those that perform the powers of x. This factor contributes to the operator having low power consumption, as will be shown later.

$$ln(1+x) = \frac{1 + 65481x - 32093x^2 + 18601x^3 - 8517x^4 + 1954x^5}{65536} \tag{8}$$

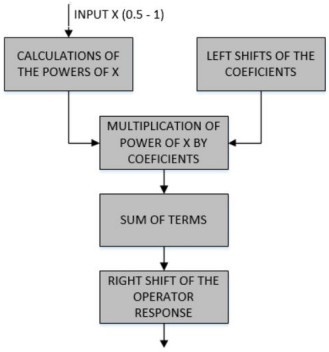

Fig. 3: Block diagram of the Taylor Series parcel.

IV. Results and Discussions

Figure 4(a) shows an error evaluation scheme using an environment linking: (a) Matlab Simulink and Modelsim tools in a co-simulation with a universal verification methodology (UVM). Generically, our error-quality verification methodology (Fig. 4c) composed of a Golden Model (GM) described in a Simulink model, and the Device Under Test (DUT) represented at the netlist level, after logic synthesis, to a standard-cell library.

Figure 4(b) also presents the proposed power estimation methodology. Firstly, a synthesis in the Cadence Genus Compiler tool was performed to generate the Verilog netlist

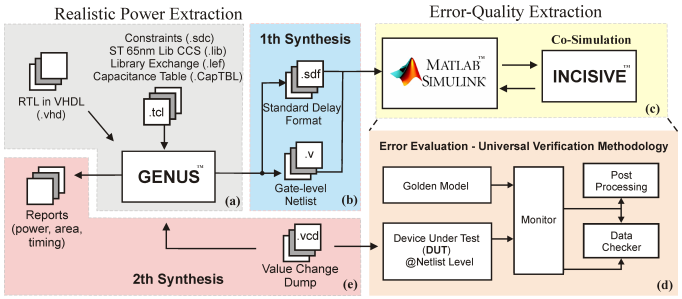

Fig. 4: Synthesis methodology diagram for realistic power extraction and co-simulation for error-quality evaluation.

and the SDF delay file. The logic simulation, which produces the VCD stimuli file, presents a testbench composed of Simulink and Modelsim, considering real input vectors used for realistic power extraction.

The methodology used to perform the tests and obtain results from the operator is the co-simulation. The co-simulating process is used to validate a system that is composed of different parts, for example, hardware and software. The co-simulation models are classified as homogeneous or heterogeneous.

Homogeneous co-simulation models throughout the system are simulated using the same tool, and this determines that the system models are specified using the same language. In heterogeneous models, it is not necessary for all entities of a project to be defined using the same language, or even the same technology. These systems can be divided into two other categories: Back-plane, which requires an environment to integrate the simulators, or Direct coupling, in which the simulators use protocols to exchange information directly.

In this work, we use a heterogeneous co-simulation model. The goal is to have parts of a system simulated using different specific tools. The simulators used in the tests are the Matlab/Simulink® version r2016a, and the ModelSim® PE Student Edition version 10.4a. Simulink® is a graphical block diagram environment used for modeling systems. It has support for simulation, automatic code generation, continuous testing and verification of embedded systems. The toolbox that has the HDL co-simulation component is called HDL Verifier. ModelSim® is a simulation environment for hardware description languages capable of performing simulations at Gate Level or Register Transfer Level (RTL). At the level of RTL the circuit is analyzed at the behavior level of the registers, and in Gate Level, it is analyzed at the netlist level with the inclusion of delays of the logical ports.

The tests performed consist of using ModelSim® to compile and simulate the codes written in VHDL and using Matlab® to perform the mathematical calculations and comparisons to determine the error. The Simulink® tool is used to integrate both simulators into the same simulation environment.

A. Simulation Results

Figure 5 shows the output curves of the Matlab log function in blue and the operator output developed in

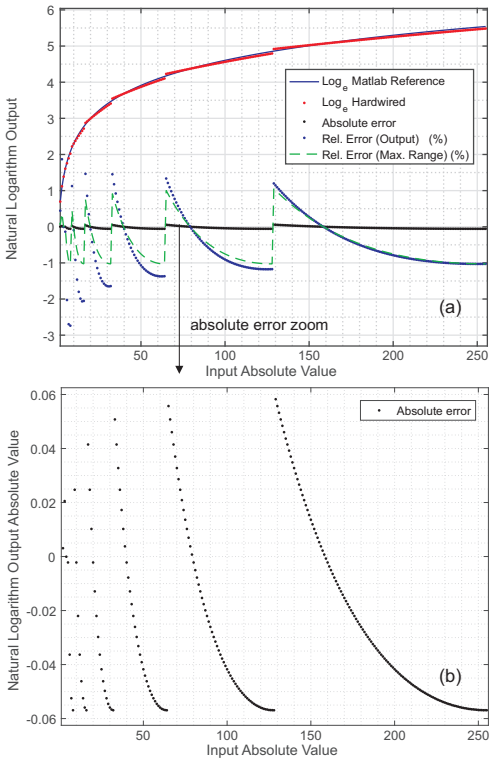

Fig. 5: Logarithmic response results (a); Zoom-in in the absolute error (b).

yellow for entries in the range $0 < x \leq 255$. The difference between these curves is shown in Figure 5.

Analyzing Figure 5-(b) one can observe that the absolute error remains below 0.0625 during the entire range of input values. At the beginning of the absolute error curve, an outbreak of magnitude 0.125 (peak-to-peak) is perceptible. It occurs due to the inherent singularity of the logarithm function when the signal input is zero.

V. Synthesis Results and Discussions

The proposed fixed-point natural logarithm arithmetic circuit was described in VHDL and synthesized for ASIC using Cadence Genus Solution for the ST Low-power CMOS 65nm technology standard-cells library at 1.0V supply, and with the maximum frequency target. The gate count is taken as the area equivalent for 2-input NANDs in the cell library. The circuits also were synthesized for FPGA using the Xilinx XC7VX1140T device target.

The synthesis tool embeds a physically-aware methodology to obtain realistic information about power dissipation and cell area. Physical characteristics of both the fabrication process and the layout of cells (contained in the LEF files) are taken into account to estimate parasitics, during logic synthesis accurately, and ahead of specific place and route. Then, each synthesis-generated netlist was simulated using random inputs to create a TCF which is fed back to the synthesis tool to calculate the power dissipation related to each circuit node.

Table II shows the circuit cell area, delay time path, maximum frequency, and total power dissipation results for the fixed-point natural logarithm arithmetic.

TABLE II: Hardware Synthesis results.

Characteristics	Hardware Results	
Device Technology	ASIC ST Bulk 65nm	FPGA Xilinx Stratix 28 nm
Circuit Area	0.7mm^2	10393 Slice Luts 2869 Slices, 340 DSPs
Frequency (MHz)	20	6.7
Static Power (mW)	0.44	67
Dynamic Power (mW)	11.00	589
Total Power (mW)	11.44	657
Energy Eff. (mJ/Mops)	0.572	9.8

For the comparison, the work presented in [8] also presented a hardwired approximation, based on the CORDIC algorithm, that can implement the natural logarithm also was synthesized for FPGA. However, the authors of [8] suppress the synthesis results for circuit area, power and maximum frequency, which makes it impossible the comparison with our solution. The work of [8] also evaluate the error from approximation, however for a small range of 4 discrete values that also impossibilities the comparison.

VI. Conclusion

This work presented a natural logarithm operator implemented in fixed-point. The results showed that the architecture introduces error less than 1%, and when the signal input is zero the output signal will also be zero, which avoids the singularity inherent of the logarithm function when the number to be calculated is zero.

References

[1] S. Shirali, *A Primer on Logarithms*, ser. Mathematical marvels. Sangam Books, 2002.

[2] C. H. Liu, O. C. Au, P. H. W. Wong, and M. C. Kung, "Image characteristic oriented tone mapping for high dynamic range images," in *2008 IEEE International Conference on Multimedia and Expo*, June 2008, pp. 1133–1136.

[3] J.-P. Deschamps, G. D. Sutter, and E. Cantó, *Guide to FPGA implementation of arithmetic functions*. Springer Science & Business Media, 2012, vol. 149.

[4] H. Anton, I. C. Bivens, and S. Davis, *Calculus Single Variable*. John Wiley & Sons, 2012.

[5] R. Gutierrez and J. Valls, "Low cost hardware implementation of logarithm approximation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 12, pp. 2326–2330, 2011.

[6] Z. Li, J. An, M. Yang, and Y. Jing, "FPGA design and implementation of an improved 32-bit binary logarithm converter," in *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM'08. 4th International Conference on*. IEEE, 2008, pp. 1–4.

[7] Helvacioğlu, Gülfem, Alp, A. B. K. Y. Kemal, Kasnakoğlu, and Coşku, "Reduced CORDIC based logarithmic convertor," in *Signal Processing and Communications Applications Conference (SIU), 2017 25th*. IEEE, 2017, pp. 1–4.

[8] L. Bangqiang, H. Ling, and Y. Xiao, "Base-N logarithm implementation on FPGA for the data with random decimal point positions," in *Signal Processing and its Applications (CSPA), 2013 IEEE 9th International Colloquium on*. IEEE, 2013, pp. 17–20.

[9] J. Lai, "Hardware Implementation of the Logarithm Function using Improved Parabolic Synthesis," 2013.

[10] H. Van Phuc, S. Van Thuan, and P. C. Kha, "LOW AREA ASIC IMPLEMENTATION OF LOGARITHM APPROXIMATION FOR DIGITAL SIGNAL PROCESSING."

[11] D. Chen, L. Han, Y. Choi, and S.-B. Ko, "Improved decimal floating-point logarithmic converter based on selection by rounding," *IEEE Transactions on Computers*, vol. 61, no. 5, pp. 607–621, 2012.