Optimizing Iterative-based Dividers for an Efficient Natural Logarithm Operator Design

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

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

***Abstract*—This work proposes to optimize iterative-based dividers for a natural logarithm operator design. The logarithm operator approximates the function at the base *e*. Such an operator is implemented using Taylor Series approximation. In this approximation, the division is a costly operation. In this work, we explore two iterative-based divider circuits for logarithm operator. We also offer an algorithm responsible for allowing input values different from the ones in the convergence region of the Taylor Series, with a reduced number of iterations. Through co-simulation using both the Matlab® and ModelSim® softwares, it was possible to determine the approximation quality of the implemented circuits, with a curve response very close to the Matlab one. Moreover, the Goldschmidt divider presents a slightly fewer relative error, mainly for higher input values comparing with Newton-Raphson. Synthesis results show that although the logarithm circuit with Newton-Raphson divider has marginally more area, it is more power-efficient than the one using Goldschmidt divider.**

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

I. Introduction

Over 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 image processing to calculate the geometric mean of the image luminance [1].

Implementations of logarithmic functions are performed using numerical approximation methods. The most commonly used approaches include Taylor Series, CORDIC, and Improved Parabolic Synthesis [2]. The work in [3] proposed the development of a logarithmic operator at the base *e* approach by Taylor Series, with the extension of the input range of the operator for entries with range $0 < x \leq 255$. In this approximation, the divider is a costly operation. This paper explores efficient iterative-based divider circuits that operate integer operands resulting in fixed-point values. One explores them in the convergence region expansion algorithm of a natural logarithm operator approximated by Taylor Series to reduce its power consumption.

The main contributions of this work are: (a) propose a circuit for calculating the initial approximation of the reciprocal of the divisor, that reduce the number of iterations in the iterative-based dividers,(b) offer an efficient divider for the logarithm architecture that significantly reduces the area and power, without changing the frequency operation.

The rest of the paper is presented as follows. The next section gives an overview of the iterative-based dividers. Section III shows the exploration of the iterative-based dividers in the logarithm operator. The main results and discussions are presented in Section IV. Finally, Section V concludes the paper.

II. Background

This section presents an overview of the natural logarithm based on the Taylor series, as well as shows the main aspects of the iterative-based dividers and shows the related work.

A. Natural logarithm based on Taylor Series approximation

The operator based on Taylor Series and the convergence region of this series is $0.5 \leq y \leq 1$. To the input receive values between 0 and 255, it is necessary to adapt the data so that they can attend the convergence in this region. Two mathematical properties of logarithms must be used to adjusts the operator input. The first is ownership of the product. This property is given in (1), where y is the value between 0.5 and 1, and c is a value that multiplied by y results in the input value of the operator.

$$\ln(x) = \ln(cy) = \ln(c) + \ln(y) \tag{1}$$

With the operation of (1), it is already possible to calculate the portion $\ln(y)$ of the equation by applying it directly to the Taylor Series. To obtain the value of y, 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 (2^k). For example, the decimal number 60, which 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. Since the implementations are in binary code, we can say that $\ln(c)$ is equal a 2^k. Then, using the power property of logarithms, the operator response is (2).

$$\ln(x) = \ln(cy) = k * \ln(2) + \ln(y) \tag{2}$$

The last step of calculating $\ln(x)$ 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 equal to 1 is in the position of the bit 2. Thus, k will be equal to 2.

Fig.1 shows the block diagram of an efficient existing logarithm hardware architecture [3]. In this work, we employ this hardware architecture to explore different divider architectures (red-colored in Fig. 1) optimized employing our proposal reciprocal approximation technique. The first step (in Fig. 1) finds the value of k and 2^k. The divider receives the 2^k as the denominator and the logarithm as the numerator.

978-1-7281-3427-7/20/$31.00 ©2020 IEEE

The next block is the Taylor Series that calculates the ln(y). Therefore the division range output is designed in the range of the Taylor series convergence region (fixed-point Q4.15). The Taylor result series is added with the K value multiplied by $\ln(2)$ to adjust the function from ln(y) to ln(x).

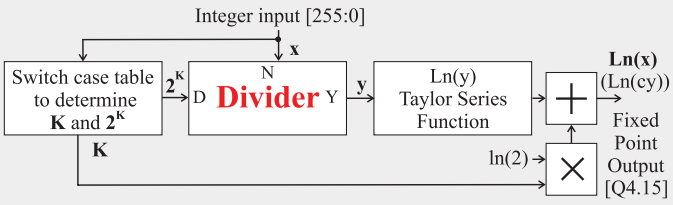

Fig. 1. Block diagram of the logarithm operator.

B. Division by functional Iteration

Division by functional iteration utilizes multiplication as the fundamental operation. Multiplicative division algorithms can take advantage concerning the digit recurrence algorithms because they can use high-speed multipliers to converge to a quadratic result [4]. This operation utilizes (3), whose Q is the quotient, N is the dividend, and D is the divisor. In this case, the challenge becomes how to compute the reciprocal of the divisor efficiently [5]. Newton-Raphson and Goldschmidt algorithms are known methods for the iterative division.

$$Q = \frac{N}{D} = N \times \frac{1}{D} \tag{3}$$

1) Newton-Raphson algorithm: This algorithm is represented in (4). In this equation, $y_{(-1)}$ is the first reciprocal approximation of the divisor, and the result is the new reciprocal value of the divisor, with more approximation of the real value. For calculating the quotient, we need utilizing (5) [4].

$$y_{(i)} = y_{(i-1)} \times (2 - D \times y_{(i-1)}) \tag{4}$$

$$Q = N \times y_{(i)} \tag{5}$$

2) Goldschmidt algorithm: This algorithm is represented in (6), (7) and (8). The first equation calculates the new value of the quotient, where the $q_{(-1)}$ is the numerator, and $e_{(-1)}$ is the first reciprocal approximation of the divisor. The second equation calculates the reciprocal of the divisor error, where the $d_{(-1)}$ is the divisor. Finally, the third equation calculates the response error.

$$q_{(i)} = q_{(i-1)} \times e_{(i-1)} \tag{6}$$

$$d_{(i)} = d_{(i-1)} \times e_{(i-1)} \tag{7}$$

$$e_{(i)} = 2 - d_{(i)} \tag{8}$$

C. Initial reciprocal approximation of the divisor

As seen previously, the Goldschmidt and Newton-Raphson algorithms need one value to an initial approximation of the reciprocal of the divisor. Frequently, this value is obtained from a lookup table. However, most works that use lookup table uses floating-point representation, as in [6], [7]. The work in [8] proposes an efficient initial approximation of the reciprocal of the integer divisor operand for a divider that results in fixed-point values. The hardware uses inputs and output on Q7.8 format and calculates the reciprocal denominator for a range of 0.0078125 to 127.99609375, according to the following steps. First, find the most significant bit position with a value equal to 1 in the integer part of the denominator (D[i])=1. After, find the fractional position (a=-(i+1)), and put the value 1 in this position (D_{opt}[a] = 1). The next position of D_{opt} receives the inversion of D[i-1], i.e., D_{opt}[a-1] = NOT D[i-1]. Following the steps of the algorithm, it is possible to generate all possible values of the initial reciprocal approximation of the divisor (D_{opt}) by the denominator value ranges. More details for the (D_{opt}) calculation can be found in [8].

D. Related Work

By reviewing works from the literature, we can find several ways of approximating the logarithm function for hardware implementation. As an example, the work in [3] uses Taylor series, logarithmic properties, and divisions to calculate the logarithm for values outside the convergence region of the Taylor series. Other methods widely used to approximate functions for calculating the logarithm is the Mitchell approximation uses to propose a low-cost implementation logarithm [9]. A CORDIC (Coordinate Rotation Digital Computer) algorithm. A CORDIC-based methodology for the computation of logarithms and exponentials with an arbitrary fixed base is presented in [10]. The work in [11] used this algorithm in their works. Based on the improved parabolic synthesis, [12] obtained results that surpassed consolidated techniques like the CORDIC and the Taylor Series. Using leading one detector and encoder (LODE) algorithm, [13] proposes an ASIC implementation for logarithm approximation. The last paper presented by [14] 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. Unlike state of the art, our work explores different iteration-based dividers dedicated to the integer logarithm hardware architecture, aiming to saving circuit area and power dissipation.

TABLE I. SUMMARY OF HARDWIRED LOGARITHM RW.

Related Work	A	B	C	D	E	F	G	H
[3]	ASIC	✓	×	$\log_e$	✓	✓	✓	✓
[8]	ASIC	✓	×	×	✓	✓	✓	✓
[9]	FPGA	×	×	$\log_2$	×	×	×	×
[10]	ASIC	✓	×	arbitrary	×	×	✓	×
[11]	FPGA	✓	×	$\log_2/log_e$	×	×	✓	×
[12]	ASIC	✓	×	$\log_2$	×	×	✓	×
[13]	ASIC	×	×	$\log_2$	×	×	✓	×
[14]	FPGA	✓	×	$\log_{10}$	×	×	×	×
This work	ASIC	✓	✓	$\log_e$	✓	✓	✓	✓

(A) Target device, (B) Integer input, Fixed-Point output
(C) Exploring Gold. and NR iterative-based dividers, (D) Logarithm bases
(E) Using initial approximation of the reciprocal
(F) Reducing the number of iterations in the dividers
(G) Power results, (H) Methodology for power results.

III. THE ITERATIVE-BASED DIVIDERS WITH THE SCHEME FOR REDUCING THE NUMBER OF ITERATIONS

We propose the implementation of a divider, with integer inputs resulting in a fixed-point output between 0 to 1. To calculate the reciprocal of the divisor, we use a method based on [8]. However, we reduced the circuit because it calculates only the integer inputs, and the result of the initial approximation of the reciprocal of the divisor is always between

0.5 and 1. To improve the accuracy of this initial value, we observe three denominator bits. If the circuit that executes this operation has W bits input, thus W+2 bits are required to the output. For the logarithm architecture of this work, the W is equal to nine. Algorithm 1 shows the proposed algorithm for choosing the initial approximation of the reciprocal of the divisor. For divisions by zero not to be performed, the result of the algorithm output is high impedance for denominator values of zero.

TABLE II. COMPARING D_{opt} WITH THE ERROR IN EACH ITERATION OF GOLDSHIMDT AND NEWTON-RAPHSON ALGORITHMS.

		Bit-Width	D_{opt} with 2 bits [8]	D_{opt} with 3 bits proposed
Numerator		8	56	56
Denominator		9	64	64
Desired Value		16	0.875	0.875
1/D Approximation		11	0.01171875	0.013671875
Goldshimdt	$D_{opt} \times N = Q_0$	11	0.6563	0.7656
	$D_{opt} \times D = D_0$	12	0.75	0.875
	2 - $D_0 = e_0$	13	1.25	1.125
	Iteration 1 (Q_1)	16	0.8203	0.8613
	Iteration 2 (Q_2)	16	0.8716	**0.8748**
	Iteration 3 (Q_3)	16	**0.8745**	0.8744
Newton-Raphson	$N \times D_{opt} = Q_0$	11	0.6563	0.7656
	$2 - D_{opt} \times D = e_0$	13	1.25	1.125
	$D_{opt} \times e_0 = y_0$	13	0.014648437	0.013183593
	Iteration 1 (Q_1)	16	0.8203	0.8613
	Iteration 2 (Q_2)	16	0.8716	**0.8733**
	Iteration 3 (Q_3)	16	**0.8733**	0.8733

Algorithm 1: The reciprocal of the divisor calculation

```
1  Parameter (W: integer:=9);
2  Input[W-1:0] Divisor
3  Output[W+1:0] Dopt
4    divisor_aux ← Divisor & "00";
5    if divisor_aux = (others → '0') do
6      Dopt ← (others→'Z');
7    else
8      for i=2 to W+1 do
9        if divisor_aux(i) = '1' do
10         Dopt ← (others → '0');
11         Dopt(W+3-i) ← '1';
12         Dopt(W+3-i-1) ← NOT (divisor_aux(i-1));
13         Dopt(W+3-i-2) ← NOT (divisor_aux(i-2));
14       end if
15     end for
16   end if
```

Table II shows an example comparing the number of iterations with both D_{opt} from the literature and our proposed D_{opt}. Using the D_{opt} algorithm proposed in [8], we can see that the best result is in the third iteration in both Goldshimdt and Newton-Raphson dividers with two bits for achieving the best D_{opt}. On the other hand, in our algorithm, we note that using one more bit in the first approximation of the denominator, we reduce one iteration, for achieving D_{opt}, without reducing the accuracy. Therefore, for both algorithm dividers, we find the final results in the second iteration ($Q2$), with our algorithm. On the other hand, in the algorithm of [8], one can obtain the final result in the third iteration ($Q3$), as can be compared in the example of Table II. Thus, we implemented the Newton-Raphson and Goldschmidt dividers using our algorithm for the initial approximation of the reciprocal of the divisor, and we employed the dividers into the logarithm architecture.

IV. RESULTS AND DISCUSSIONS

Fig. 2(d) 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. 2c) 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.

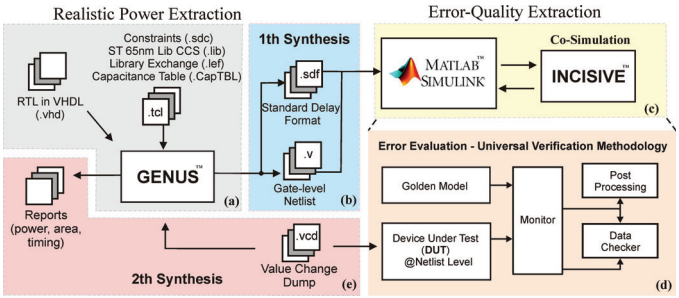

Fig. 2. Synthesis methodology diagram for realistic power extraction and co-simulation for error-quality evaluation [3].

Figs. 2(a), (b), (e) also present the proposed power estimation methodology. Firstly, a synthesis in the Cadence Genus Compiler tool was performed to generate the Verilog netlist 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.

A. Simulation results

Fig. 3 shows the output curves of the logarithm function using the Newton-Raphson and the Goldsmith dividers. We can see that the results are very similar to the model result from Matlab (called Golden Model). Fig. 4 shows the relative error from the Newton-Raphson divider, Goldschmidt divider, and the Newton-Raphson divider using in [3]. One can observe that have one error discontinuity when the denominator value of the division changes, changing the error values of the division. We are comparing the absolute error between the logarithm architecture of [3] and the proposed dividers. The Goldschmidt curve is very close to the logarithm architecture of [3], and Newton-Raphson divider inserts a little more error mainly in high denominator values, but this error is not expressive for logarithm application.

B. Synthesis results and discussions

The integer natural logarithm operator was described in VHDL. It was synthesized for ASIC using Cadence Genus Solution for the ST CMOS 65nm technology standard-cells library at 1.0V supply, and with the maximum frequency target. 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 III shows the circuit cell area, delay time path,

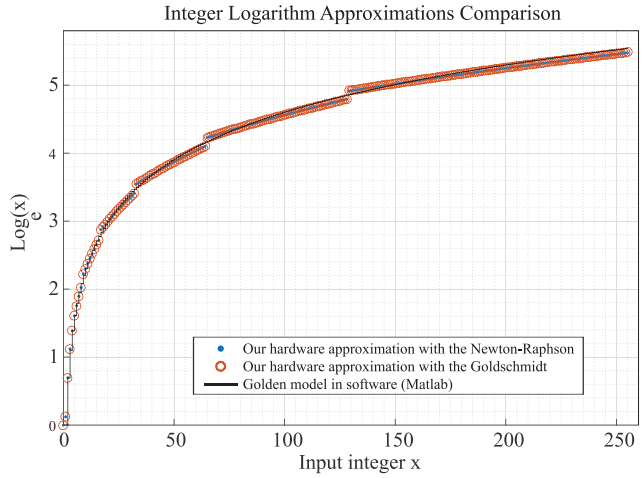

Fig. 3. Integer natural logarithm results for the golden model and the approximate architectures using Newton-Raphson and Goldschmidt dividers.

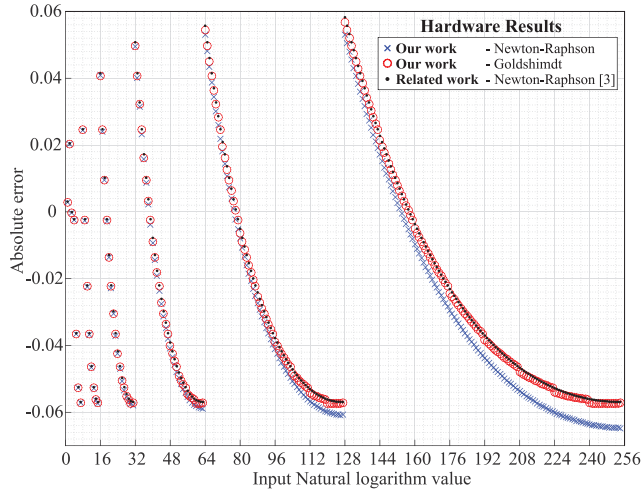

Fig. 4. Integer natural logarithm absolute error between Model vs approximate architectures using the Newton-Raphson, Goldschmidt dividers and the Newton-Raphson using in [3].

maximum frequency, and total power dissipation results for the integer natural logarithm arithmetic using Goldschmidt, Newton-Raphson dividers and natural logarithm architecture proposed in [3]. The logarithm with Newton-Raphson has a slightly more area, but it presents less total power than the one with Goldschmidt divider. It occurs because the Goldschmidt uses two multipliers in parallel at each iteration, which explains its higher dynamic power. When comparing with the logarithm circuit from the literature, we can reduce the power in 83.16% when using the Newton-Raphson divider. The significant difference in power and area occurs because the logarithm of [3] uses a Newton-Raphson divider with three iterations and 40 bits integer input, which is not required for this application. Therefore, we can conclude that the logarithm with our proposed Newton-Raphson divider is more efficient because it presents less power with a negligible error.

V. Conclusion

This work presented the exploration of Goldschmidt and Newton-Raphson iterative-based dividers in a natural logarithm circuit, with a proposed initial approximation of the reciprocal of the divisor, which reduced the number of iterations in both dividers. The logarithm implementation used the Taylor series approximation. It was possible to see that the responses of

TABLE III. Prior work comparison: Synthesis results[3,4] for the natural logarithm hardware architectures.

		This work			
	[3]	GS[1]	Savings	NR[2]	Savings
Circuit Area (mm^2)	0.70	0.031	95.57%	0.032	95.42%
Static Power (mW)	0.44	0.018	95.90%	0.019	95.68%
Dynamic Power (mW)	11.0	2.034	81.50%	1.907	82.66%
Total Power (mW)	11.44	2.052	82.06%	1.926	83.16%

Employing our optimized [1] Goldschmidt and [2] Newton-Raphson divisors proposals.

[3] Logic synthesis using the PLE (physically-aware layout estimation).

[4] Synthesised for ST 65nm at 1.0V running @ 20MHz.

the logarithm with our implemented dividers are very close to the Matlab ones. Synthesis results showed that the method proposed in this paper reduces both the area and the power for the natural logarithm application, mainly when using the Newton-Raphson divider.

References

[1] 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 *IEEE International Conference on Multimedia and Expo*, June 2008, pp. 1133–1136.

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

[3] M. R. Weirich, G. Paim, E. A. da Costa, and S. Bampi, "A Fixed-Point Natural Logarithm Approximation Hardware Design Using Taylor Series," in *2018 New Generation on CAS (NGCAS)*. IEEE, 2018, pp. 53–56.

[4] S. F. Oberman, S. F. Oberman, M. J. Flynn, and M. J. Flynn, "An Analysis Of Division Algorithms And Implementations," *IEEE Transactions on Computers*, vol. 46, pp. 833–854, 1995.

[5] S. F. Oberman and M. J. Flynn, "Division Algorithms and Implementations," *IEEE Trans. Comput.*, vol. 46, no. 8, pp. 833–854, Aug. 1997. [Online]. Available: http://dx.doi.org/10.1109/12.609274

[6] N. Louvet, J.-M. Muller, and A. Panhaleux, "Newton-Raphson algorithms for floating-point division using an FMA," in *ASAP - 21st IEEE International Conference on Application-specific Systems, Architectures and Processors*. IEEE, 2010, pp. 200–207.

[7] T. He, J. Chen, Y. Lei, Y. Peng, and B. Zhu, "High-Performance FP Divider with Sharing Multipliers Based on Goldschmidt Algorithm," *Chinese Journal of Electronics*, vol. 26, no. 2, pp. 292–298, 2017.

[8] G. Paim, P. Marques, E. Costa, S. Almeida, and S. Bampi, "Improved goldschmidt algorithm for fast and energy-efficient fixed-point divider," in *24th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2017, pp. 482–485.

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

[10] Y. Luo, Y. Wang, Y. Ha, Z. Wang, S. Chen, and H. Pan, "Generalized Hyperbolic CORDIC and Its Logarithmic and Exponential Computation With Arbitrary Fixed Base," *IEEE Tranactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 9, pp. 2156–2169, 2019.

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

[12] J. Lai, "Hardware Implementation of the Logarithm Function using Improved Parabolic Synthesis," Master Thesis Report, Lung University, Sweden, Sep. 2013.

[13] H. Van Phuc, S. Van Thuan, and P. C. Kha, "Low Area ASIC Implementation of Logarithm Approximation for Digital Signal Processing," 2016.

[14] 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.