A FIXED-POINT IMPLEMENTATION OF THE NATURAL LOGARITHM BASED ON A EXPANDED HYPERBOLIC CORDIC ALGORITHM

Daniel R. Llamocca-Obregón
llamocca.dr@pucp.edu.pe

Carla P. Agurto-Ríos
agurto.cp@pucp.edu.pe

Grupo de Procesamiento Digital de Señales e Imágenes - Pontificia Universidad Católica del Perú
Av. Universitaria s/n Cuadra 18 - Lima 32, Perú
Telf.: +511-6262000 Anexo 4681

ABSTRACT

This work presents a fixed-point hardware implementation of the natural logarithm (*ln*) function. The natural logarithm approximation is based on a expanded hyperbolic CORDIC algorithm, which allows an efficient mapping of the logarithm function onto a VLSI or FPGA architecture, since the CORDIC algorithm consists in shifts and adds. A low-cost iterative hardware is presented, from which a fast pipelined hardware can be easily obtained. Four standard bit widths are employed: 12, 16, 24 and 32. For each one, a numerical format for the input is selected and analyzed to obtain an optimum output numerical format. The set of resulting architectures were described in VHDL and were targeted to a Stratix FPGA. An error analysis has been performed for each case, with very encouraging results.

1. INTRODUCTION

The logarithm (*ln*) function is widely implemented with: Large look-up tables; and the Taylor series: a set of multiplications and adds. But these approaches use extensive hardware resources and reduce the overall performance of a system with a *ln* hardware.

Walther[1] proposed an interesting method of obtaining the *ln* function from his hyperbolic CORDIC algorithm. However, the *ln*'s limited domain attainable with this approach will not satisfy all the *ln* applications. To address this problem, Hu[2] has proposed a theoretical extension to the basic hyperbolic CORDIC algorithm, which effectively extends the domain of the *ln* function without excessively increasing the processing time and hardware, and thus satisfy all the applications of the *ln* function.

A fixed-point iterative architecture of the logarithm function based in the expanded hyperbolic CORDIC algorithm proposed by Hu[2] is presented. Results in terms of resource count and speed were obtained by targeting the architectures, described in VHDL, to a Stratix FPGA. Four standard bit widths are employed for the inputs/outputs: 12, 16, 24 and 32. For each bit width, a numerical format is selected and analyzed to obtain an optimum output format. Finally, an error analysis is performed for each numerical format by contrasting the data obtained with the fixed-point architectures with the ideal MATLAB® values.

The rest of this work is organized as follows. Section 2 describes the method of obtaining '*ln*' with the expanded hyperbolic CORDIC algorithm. Section 3 describes the architecture implemented. Section 4 presents an analysis of each bit width and the results of the FPGA implementation. Section 5 presents an error analysis. Finally, conclusions are given.

2. NATURAL LOGARITHM FROM THE HYPERBOLIC CORDIC ALGORITHM

2.1 Obtaining '*ln*' with the hyperbolic CORDIC algorithm

The original hyperbolic CORDIC algorithm, described by Walther [1], states the following iterative equations:

$$X_{i+1} = X_i + \delta_i Y_i 2^{-i} \quad Y_{i+1} = Y_i + \delta_i X_i 2^{-i}$$
$$Z_{i+1} = Z_i - \delta_i \theta_i \qquad (1)$$

Where: $\theta_i = Tanh^{-1}\left(2^{-i}\right)$, i = 1, 2, 3, … N (2)

And i is the index of the iteration. The iterations *4, 13, 40,… k, 3k + 1* must be repeated to guarantee the convergence:. To obtain *ln*, the vectoring mode is used, which defines the value of δ_i as:

$$Vectoring: \ \delta_i = -1 \quad if \ x_i y_i \geq 0, +1, otherwise \qquad (3)$$

In the vectoring mode, the quantities *X, Y* and *Z* tend to the following results, for sufficiently large *N*:

$$X_N \leftarrow A_n \sqrt{{X_0}^2 - {Y_0}^2} \qquad Y_N \leftarrow 0$$
$$Z_N \leftarrow Z_0 + Tanh^{-1}\left(Y_0 / X_0\right) \qquad (4)$$

$$\text{Where}: \ A_n \leftarrow \prod_{i=1}^{N} \sqrt{1 - 2^{-2i}} \qquad (5)$$

$$\text{Since}: \ ln(\alpha) = 2Tanh^{-1}\left(\frac{\alpha - 1}{\alpha + 1}\right), \ \alpha \in \langle 0, +\infty \rangle \qquad (6)$$

The function $ln(\alpha)$ is obtained by multiplying by 2 the final result Z_N. (Equation (4)), provided that $Z_0=0$, $X_0= \alpha+1$, *and* $Y_0= \alpha -1$.

2.2 Basic Range of Convergence of the CORDIC algorithm

The basic range of convergence [2], states the following:

$$\left| Tanh^{-1}\left(Y_0 / X_0\right) \right| \leq \theta_N + \sum_{i=1}^{N} \theta_i \qquad (7)$$

$$\rightarrow \left| Tanh^{-1}\left(Y_0 / X_0\right) \right| \leq 1.1182, \ for \ N \rightarrow \infty \qquad (8)$$

$$\rightarrow \left| Y_0 / X_0 \right|_{max} \approx 0.80694, \ for \ N \rightarrow \infty \qquad (9)$$

$$\rightarrow \left| ln(\alpha) \right|_{max} = 2 \left| Tanh^{-1}\left(\frac{\alpha - 1}{\alpha + 1}\right) \right|_{max} = 2.2364 \qquad (10)$$

Since $X_0 = \alpha+1$, and $Y_0 = \alpha-1$, then:

$$\left|\frac{\alpha-1}{\alpha+1}\right| \le 0.80694 \rightarrow 0.106843 \le \alpha \le 9.35947 \qquad (11)$$

This is the limitation of α. For α>9.35947, the algorithm is useless. As $ln(\alpha)$ grows dramatically for $\alpha \rightarrow 0$, the algorithm is also useless for values of α<0.106843.

2.3 Expansion of the Range of Convergence

The limitation of α (Eq. (11)) will not satisfy all applications of $ln(\alpha)$. Hu [2] has modified the basic hyperbolic CORDIC algorithm, by including additional iterations $(M+1)$ for negative indexes i: (i = 0, - 1, - 2, - 3, … - M)

$$\theta_i = Tanh^{-1}\left(1-2^{i-2}\right), \; for\; i \le 0 \qquad (12)$$

Therefore, the modified algorithm results:

$$For\; i \le 0 \quad \begin{cases} X_{i+1} = X_i + \delta_i\left(1-2^{i-2}\right)Y_i \\ Y_{i+1} = Y_i + \delta_i\left(1-2^{i-2}\right)X_i \\ Z_{i+1} = Z_i - \delta_i Tanh^{-1}\left(1-2^{i-2}\right) \end{cases} \qquad (13)$$

$$For\; i > 0 \quad \begin{cases} X_{i+1} = X_i + \delta_i Y_i 2^{-i} \\ Y_{i+1} = Y_i + \delta_i X_i 2^{-i} \\ Z_{i+1} = Z_i - \delta_i Tanh^{-1}\left(2^{-i}\right) \end{cases} \qquad (14)$$

X_N, Y_N,, and Z_N are as indicated in (4). δ_i is as indicated in (3). But the quantity A_n, described in (5), is redefined as follows:

$$A_n \leftarrow \left[\prod_{i=-M}^{0}\sqrt{1-\left(1-2^{(i-2)}\right)^2}\right]\left[\prod_{i=1}^{N}\sqrt{1-2^{-2i}}\right] \qquad (15)$$

The range of convergence, now becomes:

$$\left|Tanh^{-1}\left(Y_0/X_0\right)\right| \le \theta_{\max} \qquad (16)$$

$$\text{Where: } \theta_{\max} = \sum_{i=-M}^{0} Tanh^{-1}\left(1-2^{i-2}\right) + \left[Tanh^{-1}\left(2^{-N}\right) + \sum_{i=1}^{N} Tanh^{-1}\left(2^{-i}\right)\right] \qquad (17)$$

θ_{max} is the maximum value of $Z \leftarrow Tanh^{-1}$ (if $Z_0=0$), and imposes a limitation to X_0 and Y_0, and therefore to α. The values of θ_{max} are tabulated for M between 0 an 10 and shown in Table 1.

M	θ_{max}	M	θ_{max}	M	θ_{max}
0	2.09113	4	9.65581	8	22.82194
1	3.44515	5	12.42644	9	26.98070
2	5.16215	6	15.54462	10	31.48609
3	7.23371	7	19.00987		

Table 1. θ_{max} versus M for the Expanded Hyperbolic CORDIC algorithm (after Hu[2])

For example, with M = 2 ($\theta_{max}=5.16215$), the range of $Tanh^{-1}$ is [-5.16215, +5.16215]. Then $\alpha \in [3.2825x10^{-5},\; 30463.9711]$, which is a far larger domain of $ln(\alpha)$ than that of (11). It is clear that the expansion scheme does work. The more domain of $ln(\alpha)$ is needed, the more the iterations $(M+1)$ that must be executed.

3. LOW-COST ITERATIVE ARCHITECTURE

The architecture implements the *ln* function based on the expanded hyperbolic CORDIC algorithm. The inputs and outputs have the same width. As shown in Sec. 4, the intermediate registers' width and the iterations vary with the input/output width. A precision consideration [3] that extends the bit width by $ng = log_2(n)$ bits at the LSB position is used. We define:

- n: input/output bit width
- nr: width of the internal datapath (X and Y). $nr = ng + n$
- nz: width of the internal datapath for Z. $nz \ge n$
- N: number of basic iterations
- M: number of additional iterations minus one.

We define $na = nz - n$ as the additional bits that are added to the MSB part of Z, necessary as we will demonstrate in Section 4.

Fig. 1 depicts the architecture that implements the *ln* function iteratively. The two LUTs (look-up tables) store the 2 sets of angles defined in Eq. (2) and (12). A start signal begins the process. After '$M+1+N+v$' clock cycles ('v': number of repeated iterations stated in Section 2.1), the result is obtained in Z

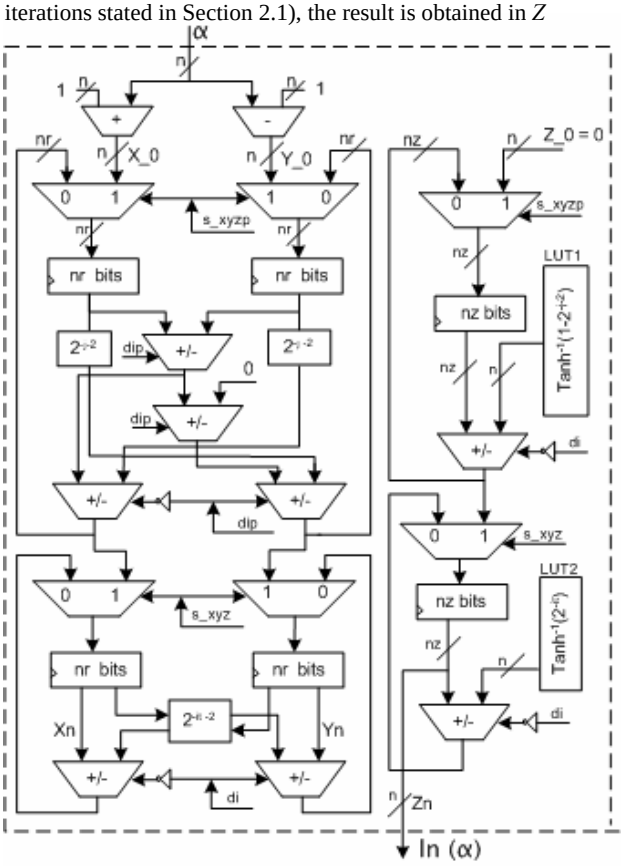

Figure 1. j = M → 0, it = 1 → N

There are 2 stages: One that performs the iterations for $i \le 0$ and is depicted in the upper part, it needs 2 muxes, 2 registers, 4 adders and two barrel shifters. This part introduces considerable delay, thus reducing the frequency of operation. The lower part of Fig. 1 implements the iterations for $i > 0$, this is a classical hardware. A state machine controls the load of the registers, the data that passes onto the muxes, the add/substract decision of the adder/substracters, and the count given to the barrel shifters.

4. ANALYSIS OF NUMERICAL FORMATS FOR EACH BIT WIDTH

Four standard bit widths for the inputs/outputs of the $ln(\alpha)$ hardware are analyzed: 12, 16, 24 and 32. In each case, an optimum architecture will be obtained, which will consider the number of normal (N) and additional iterations (M+1).

The least significant positions are always extended for $log_2(n)$ bits, where n is the input/output bit width. In the following sub-sections we will calculate the internal datapath, **but this value will not consider the guard bits ($log_2(n)$)**, because it is always present and to avoid complicating the explanation.

The numerical format is defined as: [T D], where:

T: total number of bits D: total number of fractional bits

To obtain $ln(\alpha)$, we need that $Z_0=0$, $X_0=\alpha+1$, and $Y_0=\alpha-1$.

$$\text{Then: } Z_N \leftarrow Tanh^{-1}\left(\frac{\alpha-1}{\alpha+1}\right)=\frac{ln(\alpha)}{2} \qquad (18)$$

Domain of $Tanh^{-1}$: $\langle -1,+1\rangle$. $\alpha \in \langle 0,+\infty\rangle$

The format for X and Y must be the same for correct internal operations. The maximum α attainable at each bit width defines the maximum X, and thus the format for X and Y is obtained. There is no need to add more bits to X and Y, since they tend to decrease as shown in (5). It can be shown that the smallest α causes $Z_N \leftarrow Tanh^{-1}(Y_0/X_0)$ to be maximum in absolute value, and thus, Z format is obtained. Note that X, Y and Z uses the 2's complement fixed-point fractional representation. Then we use Table 1 to find the number of additional iterations (M) needed to correctly represent Z_N (by locating the nearest θ_{max})

4.1 Input/Output bit width: 12. Format for α:[12 10] positive

$\alpha_{max} = FFFh = 3.999023 \rightarrow X_{max} = \alpha+1 = 4.999023$

With X_{max}, the format for X and Y results [12 8].

$$\alpha_{min}=001h \rightarrow Z_{N\,min}=Tanh^{-1}\left(\frac{\alpha_{min}-1}{\alpha_{min}+1}\right)=-3.465735$$

Given $Z_{N\,min}$, Z needs 3 integer bits. Thus, Z format is [12 9]. And Table 1 specifies 3 additional iterations (M=2, $\theta_{max}=5.162$).

However, in this case, the min. intermediate value for Z is -4.0439, then 1 bit must be extended to the MSB (the internal datapath for Z is 13 bits). With Z: [12 9] the LUT's angles are:

M	Value	N	Value	N	Value	N	Value
-2	36F	1	119	4	020	7	004
-1	2B5	2	083	5	010	8	002
0	1F2	3	040	6	008	9	001

Table 2

Table 2 shows that the number of iterations needed is 9. Any further iteration will yield a value less or equal than 001h for the fixed angle rotation, which is useless. In conclusion, M=2 and N=9. Z format is [12 9], and the internal datapath for Z is 13.

4.2 Input/Output bit width: 16. Format for α:[16 13] positive

$\alpha_{max} = FFFFh = 7.99987 \rightarrow X_{max} = \alpha+1 = 8.99987$

With X_{max}, the format for X and Y results [16 11].

$$\alpha_{min}=001h \rightarrow Z_{N\,min}=Tanh^{-1}\left(\frac{\alpha_{min}-1}{\alpha_{min}+1}\right)=-4.50545$$

Given $Z_{N\,min}$, Z needs 4 integer bits. Thus, Z format is [16 12]. And Table 1 specifies 3 additional iterations (M=2, $\theta_{max}=5.162$).

Since $\theta_{max}=5.162$ uses 4 integer bits, no bit will be extended for Z. With the Z format [16 12] the LUT's angles are:

M	Value	N	Value	N	Value	N	Value
-2	1B79	1	08CA	5	0080	9	0008
-1	15AA	2	0416	6	0040	10	0004
0	0F91	3	0203	7	0020	11	0002
		4	0100	8	0010	12	0001

Table 3

Table 3 shows that the number of iterations needed is 12. Any further iteration will yield a value less or equal than 001h for the fixed angle rotation, which is useless. In conclusion, M=2 and N=12. Z format is [16 12], and the internal datapath for Z is 16.

4.3 Input/Output bit width: 24. Format for α:[24 20] positive

$\alpha_{max} = FFFFFFh = 15.9999 \rightarrow X_{max} = \alpha+1 = 16.9999$

With X_{max}, the format for X and Y results [24 18].

$$\alpha_{min}=001h \rightarrow Z_{N\,min}=Tanh^{-1}\left(\frac{\alpha_{min}-1}{\alpha_{min}+1}\right)=-6.93147$$

Given $Z_{N\,min}$, Z needs 4 integer bits. Thus, Z format is [24 20]. And Table 1 specifies 4 additional iterations (M=3, $\theta_{max}=7.233$). Since $\theta_{max}=7.233$ uses 4 integer bits, no bit will be extended for Z. With the Z format [24 20] the LUT's angles are:

M	Value	N	Value	N	Value	N	Value
-3	212524	1	0F9139	6	00800B	11	000200
-2	1B78CE	2	08C9F5	7	002000	12	000100
-1	15AA16	3	04162C	8	001000	13	000080
0	0F9139	4	0202B1	9	000800	14	000040
		5	010056	10	000400	15	000020
						16	000010

Table 4

We have chosen 16 as the number of iterations. While 20 iterations can be executed, it would increase the amount of hardware excessively. In conclusion, M=3 and N=16. Z format is [24 20], and the internal datapath for Z is 24.

4.4 Input/Output bit width: 32. Format for α:[32 27] positive

$\alpha_{max} = FFFFFFFFh = 31.9999 \rightarrow X_{max} = \alpha+1 = 32.9999$

With X_{max}, the format for X and Y results [32 25].

$$\alpha_{min}=001h \rightarrow Z_{N\,min}=Tanh^{-1}\left(\frac{\alpha_{min}-1}{\alpha_{min}+1}\right)=-9.35748$$

Given $Z_{N\,min}$, Z needs 5 integer bits. Thus, Z format is [32 27]. And Table 1 specifies 5 additional iterations (M=4, $\theta_{max}=9.655$). Since $\theta_{max}=9.655$ needs 5 integer bits, no bit will be extended for Z. With the Z format [32 27] the LUT's angles are:

M	Value	N	Value	N	Value
-4	13607294	1	0464FA9F	9	00040000
-3	109291E9	2	020B15DF	10	00020000
-2	0DBC6724	3	01015892	11	00010000
-1	0AD50B1D	4	00802AC4	12	00008000
0	07C89CAC	5	00400556	13	00004000
		6	002000AB	14	00002000
		7	00100015	15	00001000
		8	00080003	16	00000800

Table 5

We have chosen 16 as the number of iterations. While 27 iterations can be executed, it would increase the amount of hardware excessively. In conclusion, M=4 and N=16. Z format is [32 27], and the internal datapath for Z is 32.

4.5 Obtaining the *ln* function from Z

The function *ln(α)* is obtained by multiplying Z_N by 2., which is tantamount to shift the fractional point 1 bit to the right, which means reading Z in the way of the third column of Table 6.:

n	Z_N format obtained in the preceding sub-sections	Z format for having *ln*
12	[12 9] (obtained in 4.1)	[12 8]
16	[16 12] (obtained in 4.2)	[16 11]
24	[24 20] (obtained in 4.3)	[24 19]
32	[32 27] (obtained in 4.4)	[32 26]

Table 6. How to read Z in order to have *ln(α)*

4.6 Results of FPGA implementation

Table 7 shows the resource effort and maximum frequency of the iterative architecture that implements the *ln* function.

Type	n	LEs	f_{max} (MHz)
ITERATIVE (Folded Recursive)	12	407	105.67
	16	525	103.77
	24	786	88.27
	32	1123	79.64

Table 7. Final Results. Device: Stratix EP1S10F484C5

The results, obtained with *Quartus II 5.0*, show that the *ln* implementation is suitable for medium and high density FPGAs.

5. ERROR ANALYSIS

For the cases analyzed in 4.1, 4.2, 4.3 and 4.4, an error analysis is performed. The results are contrasted with the ideal values obtained in MATLAB®. The error measure will be:

$$Relative\ Error = \left| \frac{ideal\ value - CORDIC\ value}{ideal\ value} \right| \qquad (19)$$

We have taken 1024 values equally spaced along the maximum domain of functions obtained for each bit width analyzed. Figures 2 and 3 show the relative error performance for the *ln (w)* function for 12, 16, 24 and 32 bits.

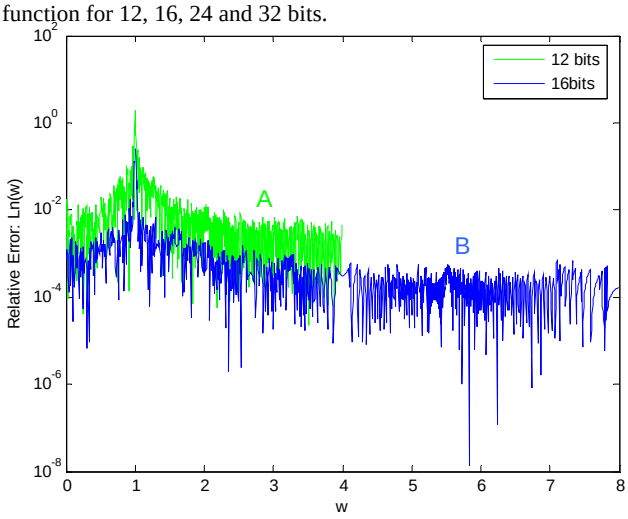

Figure 2. In Curve A, 12 bits were used ($w \in <0,4>$). In Curve B, 16 bits were used ($w \in <0,8>$).

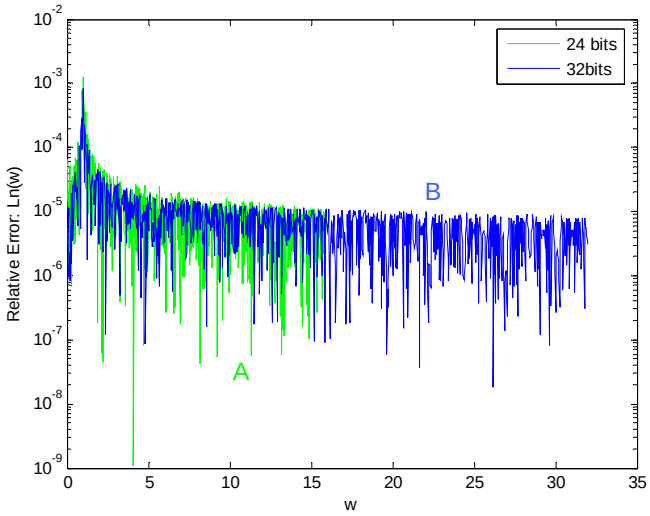

Figure 3. In Curve A, 24 bits were used ($w \in <0,16>$). In Curve B, 32 bits were used ($w \in <0,32>$).

For *w* near *1*, all the curves exhibit high relative error values, because ln(w) yields the smallest values for *w* near *1*, (note that *ln(1) = 0*)., and the fixed-point hardware fails representing those small values. This is most critical in the case of 12 bits, where the relative error is near 100% for *w* near 1. But, for 24 and 32 bits the highest relative error is approximately 0.1% for *w* near *1*.

6 CONCLUSIONS

- The architecture, has proved to be amenable for our FPGA implementation, as the clock rate and resource effort indicates. The function *ln* ha a greater domain as the bit width increases.
- We chose the format for α to be [12 10], [16 13], [24 20], and [32 27] for 12, 16, 24 and 32 bits respectively. This election is almost arbitrary. The reader can explore other alternatives, but it is our opinion that the formats chosen are the ones that work with the optimum quantity of fractional bits for each format.
- The architecture can be readily unfolded and pipelined in order to obtain a fast logarithm hardware. It is left to the reader to implement the pipelined architecture.

7 REFERENCES

[1] J.S. Walther, "A unified algorithm for elementary functions", in Proc. Spring Joint Comput. Conf., 1971, pp. 379-385.

[2] X. Hu, R. Huber, S. Bass, "Expanding the Range of Convergence of the CORDIC Algorithm", IEEE Transactions on Computers. Vol. 40, Nº 1, pp. 13-21, Jan. 1991.

[3] U. Meyer – Baese, Digital Signal Processing with Field Programmable Gate Arrays: Springer-Verlag Berlin Heidelberg, May 2001.

[4] Ray Andraka, "A survey of CORDIC algorithm for FPGA based computers".