A FIXED-POINT IMPLEMENTATION OF THE EXPANDED HYPERBOLIC CORDIC ALGORITHM

D. R. LLAMOCCA-OBREGÓN† and C. P. AGURTO-RÍOS†

† *Grupo de Procesamiento Digital de Señales, Pontificia Universidad Católica del Perú, Av. Universitaria cdra 18, Lima 32- Perú. llamocca.dr@pucp.edu.pe; agurto.cp@pucp.edu.pe*

***Abstract*— The original hyperbolic CORDIC (Coordinate Rotation Digital Computer) algorithm (Walther, 1971) imposes a limitation to the inputs' domain which renders the algorithm useless for certain applications in which a greater range of the function is needed. To address this problem, Hu *et al.* (1991) have proposed an interesting scheme which increments the iterations of the original hyperbolic CORDIC algorithm and allows an efficient mapping of the algorithm onto hardware. A fixed-point implementation of the hyperbolic CORDIC algorithm with the expansion scheme proposed by Hu *et al.* (1991) is presented. Three architectures are proposed: a low cost iterative version, a fully pipelined version, and a bit serial iterative version. The architectures were described in VHDL, and to test the architecture, it was targeted to a Stratix FPGA. Various standard numerical formats for the inputs are analyzed for each hyperbolic function directly obtained: *Sinh, Cosh, Tanh*$^{-1}$ *and exp*. For each numerical format and for each hyperbolic function an error analysis is performed.**

I. INTRODUCTION

The hyperbolic CORDIC algorithm as originally proposed by Walther (1971) allows the computation of hyperbolic functions in an efficient fashion. However, the domain of the inputs is limited in order to guarantee that outputs converge and yield correct values, and this limitation will not satisfy the applications in which nearly the full range of the hyperbolic functions is needed.

Various strategies have been proposed to address the problem of limited convergence of the hyperbolic CORDIC algorithm. One strategy is to use mathematical identities to preprocess the CORDIC input quantities (Walther, 1971). While such mathematical identities work, there is no single identity that will remove or reduce the limitations of all the functions in the hyperbolic mode. In addition, the mathematical identities are cumbersome to use in hardware applications because their implementation requires a significant increase in processing time and hardware (Hu *et al*. 1991). Another approach, proposed by Hu *et al*. (1991), involves a modification to the basic CORDIC algorithm (inclusion of additional iterations) that can be readily implemented in a VLSI architecture or in a FPGA without excessively increasing the processing time.

Three architectures for the fixed-point implementation of the hyperbolic CORDIC algorithm with the expansion scheme proposed by Hu *et al*. (1991) are presented: a low cost iterative version, a fully pipelined version, and a bit serial iterative version. Results in terms of resource count and speed were obtained by targeting the architectures, described in VHDL, to a Stratix FPGA of ALTERA®.

Four different numerical formats are proposed for the inputs. For each hyperbolic function, an analysis of each numerical format is performed and the optimal number of iterations along with the optimal format for the angle are obtained. Finally, an error analysis is performed for each hyperbolic function with each numerical format. The data obtained with the fixed-point architectures are contrasted with the ideal values obtained with MATLAB®.

II. EXPANSION SCHEME FOR THE HYPERBOLIC CORDIC ALGORITHM

A. Original Hyperbolic CORDIC algorithm

The original hyperbolic CORDIC algorithm, first described by Walther (1971), states the following iterative equations:

$$\begin{aligned} X_{i+1} &= X_i + \delta_i Y_i 2^{-i} \\ Y_{i+1} &= Y_i + \delta_i X_i 2^{-i} \\ Z_{i+1} &= Z_i - \delta_i \theta_i \end{aligned} \tag{1}$$

$$\text{Where: } \theta_i = Tanh^{-1}\left(2^{-i}\right) \tag{2}$$

And i is the index of the iteration (i=1,2,3,…N). The following iterations must be repeated in order to guarantee the convergence: *4, 13, 40,… k, 3k + 1*. The value of δ_i is either +1 or –1 depending on the mode of operation:

$$\begin{aligned} &Rotation: \quad \delta_i = -1 \quad if\ z_i < 0, +1, otherwise \\ &Vectoring: \quad \delta_i = -1 \quad if\ x_i y_i \geq 0, +1, otherwise \end{aligned} \tag{3}$$

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

$$\begin{aligned} X_n &\leftarrow A_n\left[X_0 CoshZ_0 + Y_0 SinhZ_0\right] \\ Y_n &\leftarrow A_n\left[Y_0 CoshZ_0 + X_0 SinhZ_0\right] \\ Z_n &\leftarrow 0 \end{aligned} \tag{4}$$

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

$$\begin{aligned} X_n &\leftarrow A_n\sqrt{X_0{}^2 - Y_0{}^2} \\ Y_n &\leftarrow 0 \\ Z_n &\leftarrow Z_0 + Tanh^{-1}\left(Y_0 / X_0\right) \end{aligned} \tag{5}$$

$$\text{Where '}A_n\text{' is: } A_n \leftarrow \prod_{i=1}^{N}\sqrt{1 - 2^{-2i}} \tag{6}$$

With a proper choice of the initial values X_0, Y_0, Z_0 and the operation mode, the following functions can be

directly obtained: *Sinh, Cosh, Tanh*$^{-1}$, and *exp*. Additional functions (e.g. *ln, sqrt, Tanh*) may be generated by applying mathematical identities, performing extra operations and/or using the circular or linear CORDIC algorithms (Meyer - Baese, 2001).

B. Basic Range of Convergence

The basic range of convergence, obtained by a method developed by Hu *et al*. (1991) states the following:

$$\text{Rotation Mode: } |Z_0| \le \theta_N + \sum_{i=1}^{N} \theta_i \tag{7}$$

$$\rightarrow \quad |Z_0| \le Tanh^{-1}\left(2^{-N}\right) + \sum_{i=1}^{N} Tanh^{-1}\left(2^{-i}\right) \tag{8}$$

$$\rightarrow \quad |Z_0|_{\max} \approx 1.182, \; for \; N \to \infty \tag{9}$$

This is the restriction imposed to the domain of the input argument of the hyperbolic functions in the rotation mode. Note that the domain of the functions *Sinh* and *Cosh* is <-∞,+∞>.

Vectoring Mode:

$$\left|Tanh^{-1}\left(Y_0/X_0\right)\right| \le \theta_N + \sum_{i=1}^{N} \theta_i \tag{10}$$

$$\rightarrow \quad \left|Tanh^{-1}\left(Y_0/X_0\right)\right| \le 1.1182, \; for \; N \to \infty \tag{11}$$

$$\rightarrow \quad \left|Y_0/X_0\right|_{max} \approx 0.80694, \; for \; N \to \infty \tag{12}$$

This is the limitation imposed to the domain of the quotient of the input arguments of the hyperbolic functions in the vectoring mode. Note that the domain of *Tanh*$^{-1}$ is <-1,+1>, and thus this function remains greatly limited in its domain.

C. Expansion of the Range of Convergence

The convergence range described by Eq. 9 and Eq. 12 is unsuitable to satisfy all applications of the hyperbolic CORDIC algorithm.

One strategy to address the problem of limited convergence is the use of mathematical identities to preprocess the CORDIC input quantities (Walther, 1971). However, a different preprocessing scheme is necessary for each function, making it very difficult to have a unified hyperbolic CORDIC hardware. Moreover, the preprocessing leads to a significant increase in processing time and hardware.

Hu *et al*. (1991) have proposed another scheme to address the problem of the range of convergence. The approach consists in the inclusion of additional iterations to the basic CORDIC algorithm. As it will be shown in Section III, the hardware and processing time increase is bearable and suitable for VLSI and FPGA implementation.

The method proposed by Hu *et al*. (1991) consists in the inclusion of additional iterations for negative indexes *i*:

$$\theta_i = Tanh^{-1}\left(1 - 2^{i-2}\right), \; for \; i \le 0 \tag{13}$$

Therefore, the modified algorithm results:

$$\begin{aligned} &For \; i \le 0 \\ &X_{i+1} = X_i + \delta_i\left(1 - 2^{i-2}\right)Y_i \\ &Y_{i+1} = Y_i + \delta_i\left(1 - 2^{i-2}\right)X_i \\ &Z_{i+1} = Z_i - \delta_i Tanh^{-1}\left(1 - 2^{i-2}\right) \end{aligned} \tag{14}$$

$$\begin{aligned} &For \quad i > 0 \\ &X_{i+1} = X_i + \delta_i Y_i 2^{-i} \\ &Y_{i+1} = Y_i + \delta_i X_i 2^{-i} \\ &Z_{i+1} = Z_i - \delta_i Tanh^{-1}\left(2^{-i}\right) \end{aligned} \tag{15}$$

The trend of the results for the rotation and vectoring mode is the same as that stated in Eq. 4 and Eq. 5. The value of δ_i is the same as indicated in Eq. 3. But the quantity A_n, described in Eq. 6, must be redefined as follows:

$$A_n \leftarrow \left[\prod_{i=-M}^{0} \sqrt{1 - \left(1 - 2^{(i-2)}\right)^2}\right]\left[\prod_{i=1}^{N} \sqrt{1 - 2^{-2i}}\right] \tag{16}$$

The range of convergence, stated in Eq. 7 and Eq. 10 for the basic hyperbolic CORDIC algorithm, now becomes:

$$\text{Rotation Mode: } \quad |Z_0| \le \theta_{\max} \tag{17}$$

$$\text{Vectoring Mode: } \left|Tanh^{-1}\left(Y_0/X_0\right)\right| \le \theta_{\max} \tag{18}$$

$$\text{Where: } \quad \theta_{\max} = \sum_{i=-M}^{0} Tanh^{-1}\left(1 - 2^{i-2}\right) + \left[Tanh^{-1}\left(2^{-N}\right) + \sum_{i=1}^{N} Tanh^{-1}\left(2^{-i}\right)\right] \tag{19}$$

Although Eq. 17 and Eq. 18 look nearly the same, they are interpreted differently: Equation 17 states the maximum input angle the user can enter to obtain a valid result, whereas Eq. 18 states the maximum value attainable for the *Tanh*$^{-1}$ function to which Z-Z_0 tends (according to Eq. 5). If Z_0=0, Eq. 18 states the maximum value attainable at Z, and therefore imposes a limitation to the inputs X_0 and Y_0.

The values for θ_{max} have been tabulated for M between 0 an 10 and are shown in Table 1.

For example, if M = 5 is chosen (six additional iterations), then θ_{max}=*12.42644*, and the domain of the functions *Cosh* and *Sinh* is greatly expanded to [-12.42644,+12.42644] compared with the domain in Eq. 9. Similarly, the range of the function *Tanh*$^{-1}$ is increased to [-12.42644,+12.42644], which means that the domain of the quotient Y_0/X_0 becomes nearly <-1,+1>, which is the entire domain of *Tanh*$^{-1}$.

Table 1. θ_{max} versus M for the Modified Hyperbolic CORDIC algorithm. (Hu *et al*., 1991)

M	θ_{max} from (19)	M	θ_{max} from (19)
0	2.09113	5	12.42644
1	3.44515	6	15.54462
2	5.16215	7	19.00987
3	7.23371	8	22.82194
4	9.65581	9	26.98070
		10	31.48609

From the last example, it is clear that the expansion scheme does work. The more domain of the functions is needed, the more the iterations *(M+1)* that must be executed.

III. ARCHITECTURES PROPOSED FOR THE EXPANDED HYPERBOLIC CORDIC ALGORITHM

The architectures presented here implement the expanded hyperbolic CORDIC algorithm described in Eq. 14 and Eq. 15. The architectures are such that the inputs and outputs have an identical bit width. The intermediate registers and operators can be of higher bit width due to particular details of the algorithm and precision considerations which will be explored later in this paper. As it will be shown in Section IV, the bit width of the intermediate registers, the fixed-point format of the inputs and outputs, and the number of iterations vary considerably with the input/output bit width and the particular function desired. That is, to obtain an optimum architecture which yields a particular hyperbolic function (e.g. *Tanh^{-1}, Sinh/Cosh, and exp*), the architecture has to be changed for each function and for each input/output bit width. In Section IV, we explore the particularities of each architecture for *Tanh^{-1}, Sinh, Cosh,* and *exp*.

It is worth to note, however, that a unified hyperbolic CORDIC hardware, capable of obtaining all the functions within the same architecture, is desirable for certain applications, as has been shown in Hu (1992). The same principle which will be applied to the analysis of *Tanh^{-1}, Sinh, Cosh* and *exp* can be applied to this case and thus the optimum architecture can be attained.

In addition, there exists a precision consideration which extends the bit width: it is a 'rule of thumb' found in Hu (1992): "If *n* bits is the desired output precision, the internal registers should have $log_2(n)$ additional guard bits at the LSB position". This consideration, although arbitrary, have proved to work very well.

With these considerations in mind, three fixed-point architectures are presented: a low cost iterative version, a fully pipelined version, and a bit serial iterative version. But first, it is necessary to define some nomenclature used:

n: input/output bit width
nr: bit width of the internal registers and operators
ng: additional guard bits. $ng = log_2(n)$
N: number of basic iterations
M: number of additional iterations minus one.

Note that $nr \geq ng + n$. We define the quantity $na = nr - (ng+n)$ as the additional bits that are added to the MSB part, which will be necessary as we will demonstrate in Section IV.

A. Low-Cost Iterative Architecture

Figure 1 depicts the architecture that implements the Eq. 14 and Eq. 15 in an iterative fashion. The two LUTs (look-up tables) are needed to store the two sets of elementary angles defined in Eq. 2 and Eq. 13.

The process begins when a start signal is asserted. After '$M+1+N+v$' clock cycles ('v' is the number of repeated iterations stated in Section II-A), the result is obtained in the registers *X, Y* and *Z*, and a new process can be started.

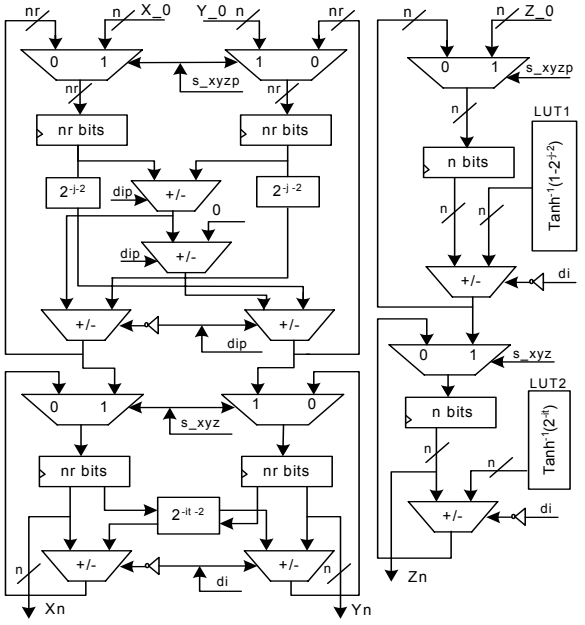

Figure 1. Iterative – CORDIC

Inputs: X_0, Y_0, and Z_0
Outputs: X_N, Y_N, and Z_N
j = M → 0 it = 1 → N

There are two stages: One that implements the iterations for $i \leq 0$ and is depicted in the upper part, it needs two multiplexers, two registers, four adders and two barrel shifters. This is the most critical part of the design, and 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 found in many textbooks and papers.

A state machine controls the load of the registers, the data that passes onto the multiplexers, the add/substract decision of the adder/substracters, and the count given to the barrel shifters.

B. Bit serial iterative architecture

The simplified interconnect and logic in a bit serial design should allow it to work at a much higher frequency than other architectures. However, the design needs to be clocked *'n'* times for each iteration (*'n'* is the width of the data). This architecture maps well in FPGA (Andraka, 1998) and is depicted in Fig. 2.

The input data (*X_0, Y_0* and *Z_0)* is loaded into the register bit per bit. Then the calculation starts. The array of serial adders/substractors, multiplexers, flip flops are arranged in a special fashion and controlled by a state machine, so that one output bit is computed every cycle, and after '$n + n*(M+1+N+v)$' cycles a new result is obtained in the registers. * 'v' is the number of repeated iterations as stated in Section II-A.

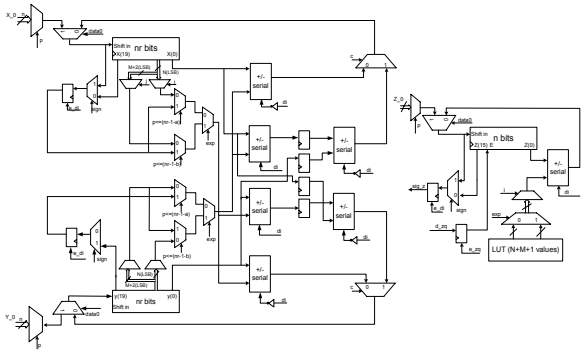

Figure 2. Bit Serial Iterative-CORDIC

C. Fully Pipelined Architecture

To develop the architecture, the algorithm described in Eq. 14 and Eq. 15 is unfolded. In addition, the stages that implement the expansion (the upper part of Fig. 2) need to be partitioned in order to avoid large delays. Therefore '2*(M+1) + N + v' stages will appear ('*v*' is the number of repeated iterations as stated in Section II-A). The architecture is depicted in Fig. 3.

Such architecture can obtain a new result each cycle. The initial latency is '2*(M+1) + N + v' cycles.

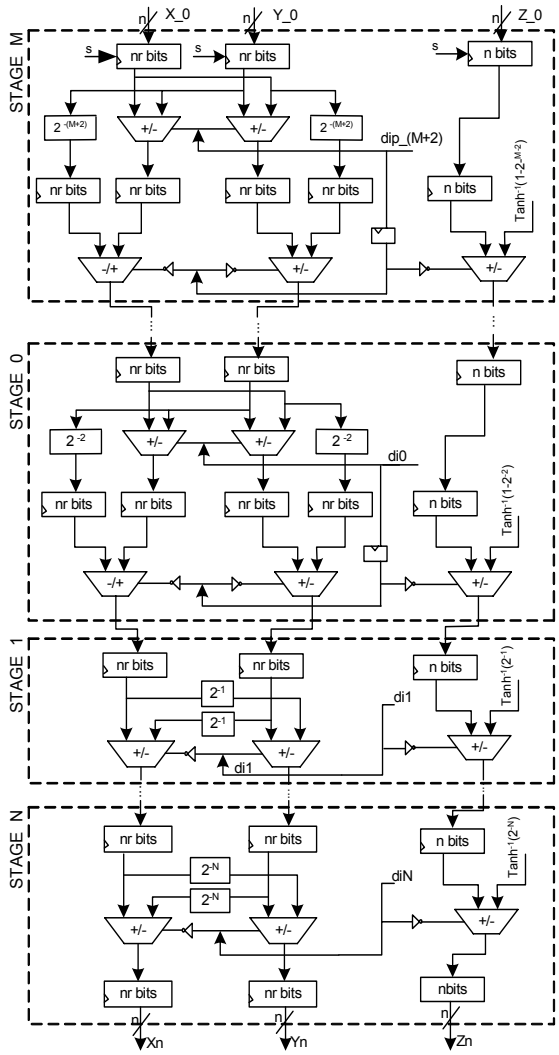

Figure 3. Pipelined-CORDIC

At each stage, *X* and *Y* have a fixed shift that can be implemented in the wiring, thus removing the barrel shifters of Section III-A. In addition, the look-up values for the θ_i are distributed as constants across the stages, which are hardwired, hence removing the look-up table. The entire hardware is reduced to an array of interconnected adder/substractors and registers. A little additional hardware in needed to obtain the '*dix*' signals, which are obtained as indicated in Eq. 3.

IV. ANALYSIS OF NUMERICAL FORMATS FOR EACH HYPERBOLIC FUNCTION

For our fixed-point hardware, 4 standard bit widths for the inputs/outputs are explored: 12, 16, 24, and 32. Our aim is to obtain an optimum architecture that implements just one hyperbolic function. As a result, we will restrict our analysis to each architecture that implements one of the following functions: $Tanh^{-1}$, *Cosh, Sinh,* and *exp*, which can be directly obtained from the hyperbolic CORDIC Eq. 14 and Eq. 15. In case a unified hyperbolic CORDIC hardware is desired, the same analysis can be applied to obtain the optimum architecture. We will show that the intermediate registers and operators need to be augmented in the MSB part and that the format for X, Y, Z, and the number of iterations varies for each architecture that implements a particular hyperbolic function. The LSB positions are always extended for *log2(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 (*log2(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.

A. Inverse Hyperbolic Tangent ($Tanh^{-1}$)

To obtain this function in *Z*, we have to set $Z_0=0$ and $X_0=1$ in the vectoring mode. Then, $Z_N \leftarrow Tanh^{-1}(Y_0)$. As the domain of $Tanh^{-1}$ is $<-1,+1>$, the input Y_0 is restricted to 1 integer bit in the 2's complement fractional fixed-point representation ($|Y_0|<1$). But, as the input $X_0=1$ requires 2 integer bits and the format for *X* and *Y* must be the same, *X* and *Y* must have 2 integer bits. The critical case occurs when Y_0 is at its max. value, from which the max. value of Z_N is obtained. 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}) . It is unnecessary to add more bits to *X* and *Y*, because they tend to decrease as shown in Eq. 5. At each bit width (12, 16, 24 and 32) we will obtain an adequate format for *Z*.

Table 2 shows the number of basic iterations (N), the additional iterations (M+1), and the set of elementary angles for each bit width. The way these values are obtained along with the format of the elementary angles is specified in the following subsections.

Input/Output bit width: 12. Format for X and Y: [12 10]

$|Y_0|_{max} = 3FFh = 0.999023475$

→ $Z_{N\ max} = Tanh^{-1}(3FFh) = 3.812065$

From Table 1, 3 additional iterations are needed (M=2, $\theta_{max}=5.162$). From Table 2, we need N=9 iterations. Given $Z_{N\ max}$, Z needs 3 integer bits.. However, the maximum intermediate value for Z is *4.04*, and 1 bit must be extended to the MSB. This change will be implemented in the internal architecture. Thus, the format for Z is [12 9] and the internal datapath is 13 bits.

Input/Output bit width: 16. Format for X and Y: [16 14]

$|Y_0|_{\ max} = 3FFFh = 0.99993896484375$

→ $Z_{N\ max} = Tanh^{-1}(3FFFh) = 5.1985885952$

Table 1 indicates that 4 additional iterations are needed (M=3, $\theta_{max}=7.23$). From Table 2, we need N=12 iterations. Given the maximum Z_N, 4 integer bits are needed to represent Z. And, as $\theta_{max}=7.23$ needs 4 integer bits, no bit will be extended. Thus, the format for Z remains [16 12] and the internal datapath is 16 bits.

Input/Output bit width: 24. Format for X and Y: [24 22]

$|Y_0|_{\ max} = 3FFFFFh$

→ $Z_{N\ max} = Tanh^{-1}(3FFFFFh) = 7.9711925$

Table 1 specifies that 5 additional iterations are needed (M=4, $\theta_{max}=9.592634$). From Table 2, we choose N=16 iterations. Given $Z_{N\ max}$, we found that 4 integer bits are needed to represent Z. However, in this case, the maxi mum intermediate value for Z is *8.5346*, so we have to extend 1 bit to the MSB. This change will be imple mented in the internal architecture. Thus, the format for Z remains [24 20] and the internal datapath is 25 bits.

Input/Output bit width: 32. Format for X and Y: [32 30]

$|Y_0|_{\ max} = 3FFFFFFFh$

→ $Z_{N\ max} = Tanh^{-1}(3FFFFFFFh) = 10.743781$

Table 2. Elementary angles for each bit width

		BIT WIDTH			
		12	**16**	**24**	**32**
-5	M				162A40FE
-4				26C0E5	13607294
-3			2125	212524	109291E9
-2		36F	1B79	1B78CE	0DBC6724
-1		2B5	15AA	15AA16	0AD50B1D
0		1F2	0F91	0F9139	07C89CAC
1	N	119	08CA	0F9139	0464FA9F
2		083	0416	08C9F5	020B15DF
3		040	0203	04162C	01015892
4		020	0100	0202B1	00802AC4
5		010	0080	0010056	00400556
6		008	0040	00800B	002000AB
7		004	0020	002000	00100015
8		002	0010	001000	00080003
9		001	0008	000800	0040000
10			0004	000400	00020000
11			0002	000200	00010000
12			0001	000100	00008000
13				000080	00004000
14				000040	00002000
15				000020	00001000
16				000010	00000800

Table 1 specifies that 6 additional iterations are needed (M=5, $\theta_{max}=12.42644$). From Table 2, we choose N=16 iterations. Given the maximum Z_N, 5 integer bits are needed to represent Z. And, as $\theta_{max}=12.42644$ requires 5 bits, no bit is needed to be extended. Thus, the format for Z remains [32 27]. The internal datapath is 32 bits.

We have chosen the number of iterations to be 16 for 24 and 32 bits. While 20 iterations can be executed for 24 bits, and 27 iterations for 32 bits, it would increase the amount of hardware excessively.

B. Hyperbolic Sine and Hyperbolic Cosine

To obtain these functions in X and Y, it is necessary to set $Y_0=0$ and $X_0=1/A_n$ in the rotation mode. Then, $Y_N \leftarrow Sinh(Z_0)$ and $X_N \leftarrow Cosh(Z_0)$. As the domain of *Sinh* and *Cosh* is $<-\infty,+\infty>$, there is no input restriction. Our strategy will consist in fixing to [12 10] the input Z for the bit width of 12, and increment 1 integer bit for a larger bit width, so that by augmenting the bit width, the range of the functions *Sinh and Cosh* is incremented.

The critical case occurs when $|Z_0|$ is at the max. value attainable at each format, from which the max. values of X_N and Y_N are obtained. Then we use Table 1 to find the number of additional iterations (M) needed to correctly represent Z_0. There is no need to add more bits to Z, because Z tends to 0 as shown in Eq. 4. At each different bit width (12, 16, 24 and 32) we will obtain an adequate format for X and Y.

Table 3. Elementary angles for each bit width

		BIT WIDTH			
		12	**16**	**24**	**32**
-5	M				162A40FE
-4				26C0E5	13607294
-3				212524	109291E9
-2			36F2	1B78CE	0DBC6724
-1			2B54	15AA16	0AD50B1D
0		3E4	1F22	0F9139	07C89CAC
1	N	232	1194	0F9139	0464FA9F
2		106	082C	08C9F5	020B15DF
3		081	0405	04162C	01015892
4		040	0201	0202B1	00802AC4
5		020	0100	010056	00400556
6		010	0080	00800B	002000AB
7		008	0040	002000	00100015
8		004	0020	001000	00080003
9		002	0010	000800	00040000
10		001	0008	000400	00020000
11			0004	000200	00010000
12			0002	000100	00008000
13			0001	000080	00004000
14				000040	00002000
15				000020	00001000
16				000010	00000800

Table 3 shows the number of basic iterations (N) and the additional iterations (M+1) necessary for each bit width. The way these values are obtained is shown in the following subsections. In addition, it also shows the set of elementary angles needed for each bit width. The

format of these angles and the procedure to obtain these formats are specified in the following subsections.

Input/Output bit width: 12. Input format for Z: [12 10]
→ $|Z_0|_{max} = |400h| = |-2|$
From Table 1, we need 1 additional iteration (M=0). From Table 3, we need N=10 iterations. Then, $A_n=0.547776019905$ and $X_0=1/A_n=1.82556366774193$. Also, the maximum values of X_N and Y_N, given $|Z_0|_{max}$ are: $X_N = Sinh(400h) = -3.62686040784702$
$Y_N = Cosh(400h) = +3.76219569108363$

Given these maximum values, 3 integer bits are needed to represent *X* and *Y*. In addition, the maximum intermediate values for *X* is *2.510150043145* and for *Y* is *2.281954584677*. So, no bit needs to be extended. Thus the format for X and Y remains [12 9] and the internal datapath for X and Y is 12 bits.

Input/Output bit width: 16. Input format for Z: [16 13]
→ $|Z_0|_{max} = |8000h| = |-4|$
From Table 1, we need 3 additional iterations (M=2). Table 3 shows that N=13 iterations are needed. Any further iteration will yield a value less or equal than 001h for the fixed angle rotation, which is useless. Then, $A_n = 0.09228252133203$ *and* $X_0 = 1/A_n = 10.83628823276322$. Also, the maximum values of X_N and Y_N, given $|Z_0|_{max}$ are:
$X_N = Sinh(8000h) = -27.289917197$
$Y_N = Cosh(8000h) = +27.3082328361$

These maximum values indicate that 6 integer bits are needed to represent *X* and *Y*. In addition, in this case, the maximum intermediate values for *X* is *34.45601024011* and for *Y* is *-34.4348456146*, so we have to extend 1 bit to the MSB. This change will be implemented in the internal architecture. Thus, the format for *X* and *Y* remains [16 10] and the internal datapath for X and Y is 17 bits.

Input/Output bit width: 24. Input format for Z: [24 20]
→ $|Z_0|_{max}=|800000h|=|-8|$
From Table 1, 5 additional iterations are needed (M=4). With the *Z* format [24 20] the $\theta_i s$ defined for the LUT are those of Table 3, from which we have chosen N=16 basic iterations. While 20 iterations can be executed, it would increase the amount of hardware excessively. Then, $A_n=4.0305251x10^{-3}$ and $X_0=1/A_n=248.1066$. Also, the maximum values of X_N and Y_N, given $|Z_0|_{max}$ are: $X_N = Sinh(800000h) = -1490.47882$
$Y_N = Cosh(800000h) = +1490.47916$

Given these maximum values, we found that 12 integer bits are needed to represent correctly *X* and *Y*. In addition, in this case, the maximum intermediate values for *X* is *3081.0854* and for *Y* is *3081.085173*, so we have to extend 1 bit to the MSB. This change will be implemented in the internal architecture. Thus, the format for *X* and *Y* remains [24 12] and the internal datapath for *X* and *Y* is 25 bits.

Input/Output bit width: 32. Format for Z: [32 27]
→ $|Z_0|_{max} = |80000000h| = |-16|$

Table 1 states that 8 additional iterations are needed (M=7). With the Z format [32 27] the $\theta_i s$ defined for the LUT are those of Table 3. We have chosen N=16 basic iterations. While 27 iterations can be executed, it would increase the amount of hardware excessively. Then, $A_n = 2.7737x10^{-6}$ and $X_0 = 1/A_n = 3.605287519x10^{-5}$. Also, the maximum values of X_N and Y_N, given $|Z_0|_{max}$ are:
$X_N = Sinh(80000000h) = 4.44305526x10^6$
$Y_N = Cosh(80000000h) = 4.44305526x10^6$
Given these maximum values, 24 integer bits are needed to represent *X* and *Y*. In addition, the max. intermediate value for *X* is $20.32x10^6$ and for *Y* is $20.32x10^6$, so we have to extend 2 bits to the MSB. This change will be implemented in the internal architecture. Thus, the format for *X* and *Y* remains [32 8] and the internal datapath is 34 bits. The format for *Z*([32 27]) is a good format, since with [32 26], we would need more than 32 integer bits for X and Y, that is impossible to implement. In conclusion, M=7 and N=16. *X, Y* format is [32 8]. The internal datapath for *X, Y* is 34.

C. Exponential (e^x)

To obtain this function, we have to set $X_0=Y_0=1/A_n$ in the rotation mode. Then, $Y_N \leftarrow Sinh(Z_0) + Cosh(Z_0)$ and $X_N \leftarrow Cosh(Z_0)+Sinh(Z_0)$. As $e^w = Sinh(w) + Cosh(w)$, we can rewrite: $Y_N \leftarrow e^{Z0}$ and $X_N \leftarrow e^{Z0}$. The domain of e^w is $<-\infty,+\infty>$, hence there is no input restriction. Our strategy will consist in fixing to [12 10] the input *Z* for the bit width of 12, and incrementing 1 integer bit for each larger bit width, so that by augmenting the bit width, the range of the function e^w is incremented. It is worth to mention that the hardware is identical to the hardware that computes *Sinh* and *Cosh*. The critical case occurs when Z_0 is the max. value attainable at each format, from which the max. values of X_N and Y_N are obtained. Then we use Table 1 to find the number of additional iterations (*M*) needed to represent Z_0. As *Z* tends to 0 (as shown in Eq. 4), it is needless to add more bits to *Z*. At each different format (12, 16, 24 and 32) we obtain an adequate format for the bits of *X* and *Y*.

The analysis is similar to the case of section IV-B, and since the maximum value for X and Y will be $X_N=Y_N\approx 2\cosh(Z_0)$, we found that 1 additional integer bit is needed, which means that the formats obtained in 4.2 will lose 1 fractional bit and X and Y must be read differently (12 bits: X, Y format = [12 8]; 16 bits: X,Y format = [16 9]; 24 bits: X, Y format = [24 11]; 32 bits: X, Y format = [32 7]).

D. Results of FPGA implementation

Note that the hardware for obtaining *exp and Sinh/Cosh* is exactly the same, though the results are interpreted differently.

The results, obtained with *Quartus II 5.0*, show that the hyperbolic CORDIC implementation is amenable to FPGA. The clock rates are relatively high and the resource effort is bearable for high-density FPGAs.

V. ERROR ANALYSIS

For the cases analyzed in sections IV-A, IV-B and IV-C, an error analysis is performed. The results are con

Table 4. Final Results - Stratix EP1S10F484C5

Type	N	Function	LEs	f_{max}
Iterative (Folded Recursive)	12	Sinh/Cosh, exp	402	97.53
		$Tanh^{-1}$	433	108.75
	16	Sinh/Cosh, exp	544	85.73
		$Tanh^{-1}$	516	99.70
	24	Sinh/Cosh, exp	866	77.50
		$Tanh^{-1}$	869	84.74
	32	Sinh/Cosh, exp	1170	83.03
		$Tanh^{-1}$	1119	87.92
Fully Pipelined (Unfolded Pipelined)	12	Sinh/Cosh, exp	662	182.22
		$Tanh^{-1}$	675	187.18
	16	Sinh/Cosh, exp	1308	168.75
		$Tanh^{-1}$	1396	177.34
	24	Sinh/Cosh, exp	1512	160.35
		$Tanh^{-1}$	1527	171.15
	32	Sinh/Cosh, exp	1686	152.35
		$Tanh^{-1}$	1718	165.40
Bit Serial Iterative (Folded recursive)	12	Sinh/Cosh, exp	225	206.10
		$Tanh^{-1}$	234	208.77
	16	Sinh/Cosh, exp	345	192.74
		$Tanh^{-1}$	348	193.25
	24	Sinh/Cosh, exp	469	182.50
		$Tanh^{-1}$	478	188.62
	32	Sinh/Cosh, exp	703	187.40
		$Tanh^{-1}$	710	191.50

trasted with the ideal values obtained in MATLAB®. The error measure will be:

$$Relative\ Error = \left| \frac{ideal\ value - CORDIC\ value}{ideal\ value} \right| \quad (20)$$

The three cases will be tested. We have taken 1024 values equally spaced along the maximum domain of functions obtained for each bit width analyzed. In the case of $Tanh^{-1}$, it has been necessary to add more values, for the $Tanh^{-1}$ grows dramatically as its argument nears ± 1.

A. Inverse Hyperbolic Tangent

We will show the relative error for the hardware that implements the hyperbolic tangent in its entire domain for 12, 16, 24 and 32 bits. Figures 7, 8, and 9 show the relative error performance for the function $Tanh^{-1}(w)$ for 12, 16, 24 and 32 bits. Although the domain of $Tanh^{-1}$ is <-1,+1>, we have just plotted for $w \in [0,+1>$ since $Tanh^{-1}$ is an odd function.

For w near 0, all the curves exhibit high relative error values, because $Tanh^{-1}(w)$ yields the smallest values for w near 0, and the fixed-point hardware fails representing those small values.

The more the bit width, the less the relative error. For example, for 12 bits (Fig. 4) nearly all the relative error values are below 10^{-2} (an error below 1%), and for 24 bits (Fig. 5), the relative error values are below 10^{-4} (an error below 0.1%).

The curve for 32 bits (Fig. 6) exhibits some irregularities due to the reduced basic iterations (16); but in general it provides the least relative error. However, it is unusual to have a $Tanh^{-1}$ hardware with a bit input data width of 32 bits.

B. Hyperbolic Sine and Hyperbolic Cosine

We will show the relative error for the hardware that implements *Sinh* and *Cosh* in the maximum domain obtained in 4.2 for 12, 16, 24 and 32 bits.

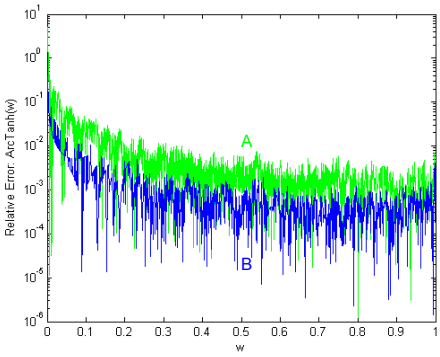

Figure 4. In Curve A, 12 bits were used. In Curve B, 16 bits were used.

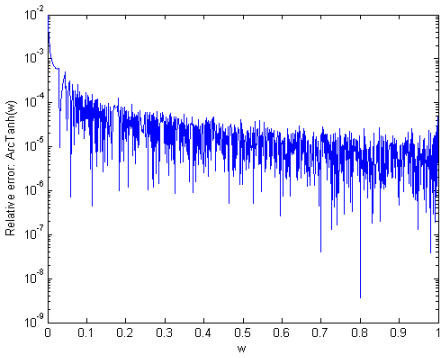

Figure 5. In the curve, 24 bits were used.

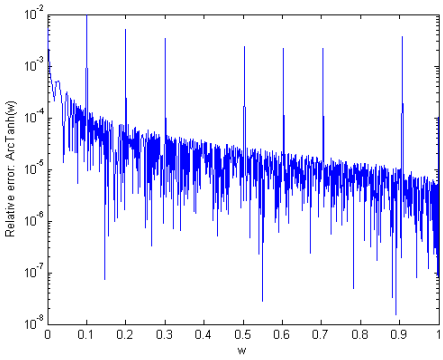

Figure 6. In the curve, 32 bits were used.

Figures 7 and 8 show the relative error for *Cosh(w)* for 12, 16, 24 and 32 bits. We have just plotted the positive domain. The negative domain is not plotted, since *Cosh* is an even function and will yield the same values.

The curve A (Fig. 8) for 24 bits is very regular because in this format we use a larger quantity of fractional bits than with the other formats.

The curve B (Fig. 8) for 32 bits exhibits some irregularities due to the reduced fractional bits (8) and the reduced number of basic iterations (16). However, it

provides the greatest domain for the *Cosh(w)* function ($w \in [-16,16\rangle$).

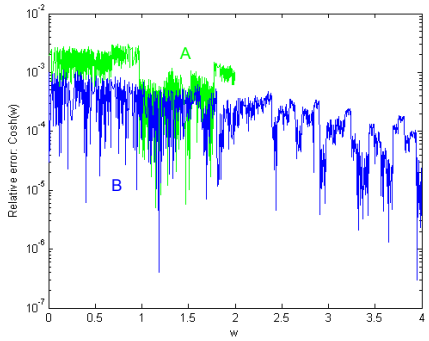

Figure 7. In Curve A, 12 bits were used (w ∈ [0,2>).
In Curve B, 16 bits were used (w ∈ [0,4>).

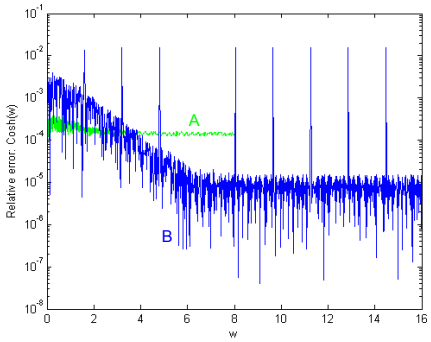

Figure 8. In Curve A, 24 bits were used ($w \in [0,8>$).
In Curve B, 32 bits were used ($w \in [0,16>$).

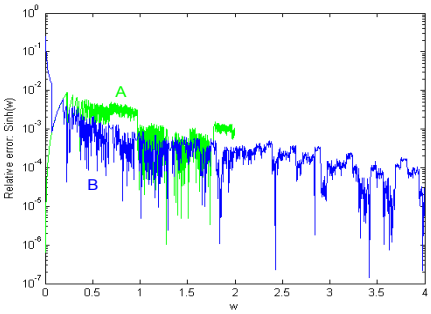

Figure 9. In Curve A, 12 bits were used($w \in [0,2>$).
In Curve B, 16 bits were used($w \in [0,4>$).

Figures 9 and 10 show the relative error performance for *Sinh(w)* for 12, 16, 24 and 32 bits. We have just plotted the positive domain. The negative domain is not plotted, since *Sinh* is an odd function and will yield the negative values of those obtained in the positive domain.

The curve A (Fig. 10) for 24 bits is very regular because in this format we use a larger quantity of fractional bits than with the other formats.

The curve B (Fig. 10) for 32 bits exhibits some irregularities due to the reduced fractional bits (8) and the reduced number of basic iterations (16). However, it provides the greatest domain for the *Sinh(w)* function ($w \in [-16,+16>$).

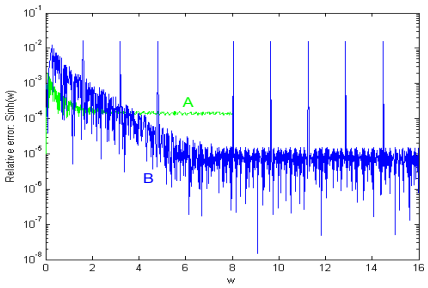

Figure 10. In Curve A, 24 bits were used ($w \in [0,8>$).
In Curve B, 32 bits were used ($w \in [0,16>$).

C. Exponential

We will show the relative error for the hardware that implements e^x in the domain obtained in section IV-C for 12, 16, 24 and 32 bits. Figures 11 and 12 show the relative error for e^w for 12, 16, 24 and 32 bits.

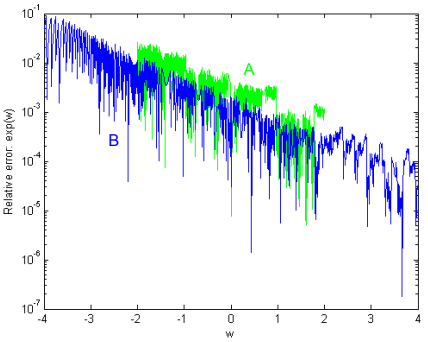

Figure 11. In Curve A, 12 bits were used($w \in [-2,2>$).
In Curve B, 16 bits were used($w \in [-4,4>$).

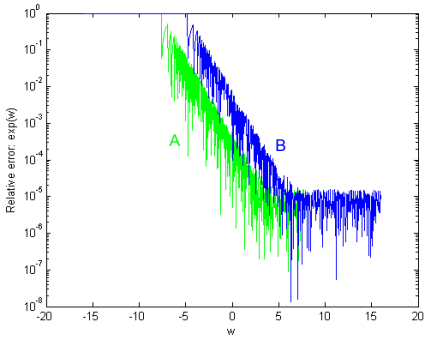

Figure 12. In Curve A, 24 bits were used($w \in [-8,8>$).
In Curve B, 32 bits were used($w \in [-16,16>$).

Note that, as *w* is more negative, the error increases and even becomes constant (as in Fig. 12). The reason is that e^w is very small for large negative values of *w*, and the fixed-point hardware fails representing those small values.

VI. CONCLUSIONS

The expansion scheme proposed by Hu *et al*. (1991), despite the additional hardware needed, has proved to be amenable for our FPGA implementation, as the clock rate and resource effort indicates. The function *Tanh-1* gets expanded in its entire domain and the functions

Cosh and *Sinh* have a greater domain as the bit width increases.

The analysis for a unified CORDIC algorithm has not been performed in order to not to lengthen this paper. But the analysis for this case is very similar to that of Section IV.

The error analysis shows certain irregularities in the relative error performance. These irregularities are due to the truncation of the fractional bits and the ever-limited number of basic and additional iterations. We have tested the CORDIC algorithm in MATLAB® and have found that the error performance is uniform.

REFERENCES

Andraka, R., "A survey of CORDIC algorithm for FPGA based computers", *Proceedings of the ACM/SIGDA*, 191-200 (1998).

Hu, X., R. Huber and S. Bass, "Expanding the Range of Convergence of the CORDIC Algorithm", *IEEE Transactions on Computers,* **40**, 13-21 (1991).

Hu, Y., "CORDIC-Based VLSI Architectures for Digital Signal Processing*", IEEE Signal Processing Magazine,* **9**, 16-35 (1992).

Meyer–Baese, U., *Digital Signal Processing with Field Programmable Gate Arrays*, Springer-Verlag Berlin, Heidelberg (2001).

Walther, J.S., "A unified algorithm for elementary functions", *Proc. Spring Joint Comput. Conf*., **38**, 379-385 (1971).

Received: April 14, 2006.
Accepted: September 8, 2006.
Recommended by Special Issue Editors Hilda Larrondo, Gustavo Sutter.