International Journal of Electrical, Electronic and Communication Sciences
ISSN: 2517-9438
Vol:7, No:12, 2013

A Multi Cordic Architecture on FPGA Platform

Ahmed Madian, Muaz Aljarhi

***Abstract*—**Coordinate Rotation Digital Computer (CORDIC) is a unique digital computing unit intended for the computation of mathematical operations and functions. This paper presents A multi CORDIC processor that integrates different CORDIC architectures on a single FPGA chip and allows the user to select the CORDIC architecture to proceed with based on what he wants to calculate and his needs. Synthesis show that radix 2 CORDIC has the lowest clock delay, radix 8 CORDIC has the highest LUT usage and lowest register usage while Hybrid Radix 4 CORDIC had the highest clock delay.

***Keywords*—**Multi, CORDIC, FPGA, Processor.

I. Introduction

COORDINATErotation digital computeris a unique digital computing unit intended for the computation of mathematical operations and functions. Initially, it was made by Volder [1], [2] for trigonometric functions which were defined in terms of main plane rotations

Walther [3], [4] has put forward an integrated procedure for the CORDIC algorithm to compute functions in circular, linear, and hyperbolic coordinate systems. From then on, CORDIC has gained more interest due to its potential for efficient and low cost implementation of a wide class of applications and became the choice for scientific calculator applications. Now, CORDIC is used in a wide variety of applications such as digital signal processing, linear transformations, digital filters, biomedical signal processing, and neural networks.

The rest of the paper is organized as follows. In Section II, brief background on unified CORDIC algorithm is presented. Section III describes related works about CORDIC algorithms and architectures. Section IV gives some investigation into the Multi CORDIC Algorithm and methods that where applied. Section V presents an implementation of a multi CORDIC processor on FPGA. Section VI gives experimental results. Section VII concludes this paper.

II. Background

The generalized iteration equations of the CORDIC algorithm are defined as [5]:

$$\begin{cases} x_{i+1} = k_i(x_i - m\sigma_i y_i r^{-S(m,i)}) \\ y_{i+1} = k_i(y_i + \sigma_i x_i r^{-S(m,i)}) \\ z_{i+1} = z_i - \sigma_i \alpha_{m,i} \end{cases} \quad (1)$$

where the coordinate parameter m defines the coordinate system (circular for m = 1, linear for m = 0, and hyperbolic coordinate for m = -1), r represents the radix of the number system, $\sigma_i$is the rotation direction (clockwise or counter clockwise), $\alpha_{m,i}$ is the rotation angle, $S_{m,i}$ is the integer shift sequence and $k_i$is the scaling factor.

TABLE I
Outputs of the Cordic Algorithm

Coordinate	Rotation($Z_n \rightarrow 0$)	Vectoring($(Y_n \rightarrow 0$)
Circular (m = 1)	$x_n = \frac{1}{K_m}(x*\cos z - y*\sin z)$ $y_n = \frac{1}{K_m}(y*\cos z + x*\sin z)$	$x_n = \frac{1}{k_m}(x^2 + y^2)^{1/2}$ $z_n = z + \tan^{-1}(y/x)$
Linear (m = 1)	$x_n = x$ $y_n = y + x * z$	$x_n = x$ $z_n = z + y/x$
Circular (m = 1)	$x_n = \frac{1}{K_m}(x*\cosh z + y*\sinh z)$ $y_n = \frac{1}{K_m}(y*\cosh z + x*\sinh z)$	$x_n = \frac{1}{k_m}(x^2 - y^2)^{1/2}$ $z_n = z + \tanh^{-1}(y/x)$

The Cordic algorithm can be implemented in two different modes, the rotation mode and the vectoring mode. The rotation mode is used to perform the rotation of a vector (x,y) by a given angle θ. The vectoring mode computed the angle θ of a vector and its magnitude by performing a finite number of rotations. In rotation mode σ_i = sign(z_i) and z_i is driven iteratively to 0 while in vectoring mode σ_i = -sign(y_i) and y_i is driven iteratively to 0 [5].

The necessary rotations are not ideal and increase the magnitude of the vector. In order to retain a constant vector length, the obtained results have to be scaled by the scale factor [5]:

$$K_m = \prod_i k_i = \prod_i \sqrt{1 + m\sigma_i^2 r^{-2S(m,i)}}$$

The functions, directly computed by the CORDIC algorithm, according to the selected value of the mode parameter and coordinate parameter, are summarized in Table I. With the appropriate initial values of x, y, and z, several basic functions can be computed.

III. Related Works

A. Most Used CORDIC Algorithms

The most well known and used CORDIC algorithms are the radix-2 and the radix-4 CORDIC algorithms. The main disadvantages of using the radix-2 CORDIC algorithm are its relatively high latency and its low throughput due to the sequential nature of the iteration process which includes carry propagation and variable shifting in every iteration. To overcome these disadvantages, pipelined implementations where proposed [6]. However, carry propagation still hinders further throughput improvement. To increase the speed of CORDIC implementation, two methods were proposed. One

A. Madian and M. Aljarhi are with the Electronics Department, German University in Cairo (e-mail: Ahmed.madian@guc.edu.eg, Muaz.al.jarhi@gmail.com).

involves reducing the delay of each iteration by using redundant arithmetic [7], [8], [10], [11], [25], [27] to remove carry propagation. The other method reduces the number of iterations by using a higher radix in the CORDIC algorithm implementation [5], [12]-[14], [19], [20], [32].

Radix-4 CORDIC algorithm offers less iterations (n/2) than the radix-2 CORDIC algorithm. Although the iterations are halved, the Radix-4 z-path involves the computation of estimated z_i and evaluation of selection function to determine σ_i resulting in higher iteration delay compared to that of radix-2 [5].

The scale factor compensation methodology involves scaling of the final (x_n,y_n) coordinates with 1/K. The natural way to do it is by using the CORDIC module in linear mode but its computational effort is the same as the CORDIC algorithm itself. Since 1/K is constant for radix 2, the computational cost can be reduced by using a CSD constant multiplier. Moreover, scaling can also be implemented using a Wallace tree by fully parallelizing multiplication and is preferred for applications aiming for low latency at the expense of more silicon area [5].

The CORDIC algorithm involves the rotation of a vector to reduce the z or y coordinates of the final vector as closely as possible to zero for rotation or vectoring mode respectively. The maximum value of the rotation angle by which the vector can be rotated depends on the shift order. The supposed results of the CORDIC algorithm can be achieved if the z or y coordinate is driven sufficiently close to zero. This can be guaranteed if the initial values of the input vector (x, y, z) lies within acceptable ranges depending on the coordinate mode m. These ranges define the domain of convergence of the CORDIC algorithm [5].

The precision of the CORDIC algorithm is affected by two kinds of computation errors explicitly, angle approximation and rounding error. The error ranges for these two sources of error are driven by performing detailed numerical analysis of the CORDIC algorithm. For angle approximation, one bit of precision is usually produced per iteration while for rounding; a maximum of $\log_2 n$ error is produced as the result of truncation of immediate results after each iteration. The Angular error and the rounding error derived are combined to yield an overall quantization error in the CORDIC implementation. The overall error can be guaranteed to be within range by considering an additional $\log_2 n$ guard bits in the implementation of the CORDIC algorithm [5].

B. CORDIC Architectures

There are two architectures for implementing CORDIC in hardware, mainly the folded and unfolded architectures. Folded architectures are acquired by reproducing each of the different iterative equations of the CORDIC algorithm into hardware and time multiplexing all the iterations into a single functional unit as shown in Fig. 1. These can be further categorized into bit-serial and word-serial architectures depending on whether the functional unit implements the logic for one bit or one word of each iteration of the CORDIC algorithm. Unfolded architectures are acquired by unfolding the iteration process so that each processing element always perform at the same iteration as shown in Fig. 2 [5], [28]-[30].

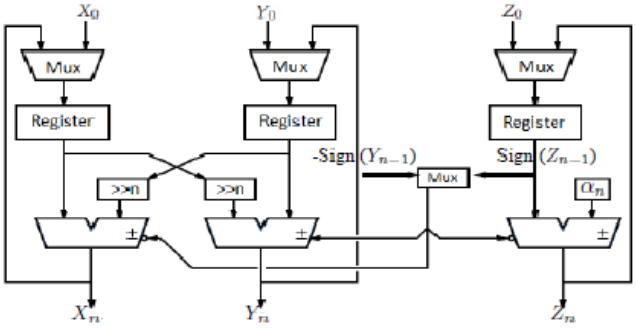

Fig. 1 Folded Iterative CORDIC Architecture

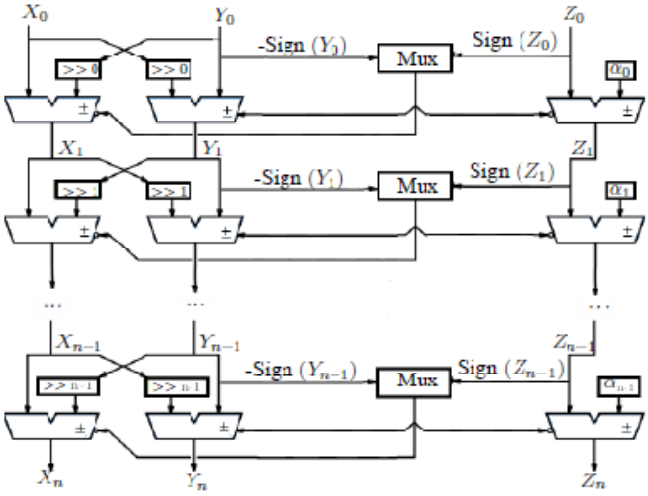

Fig. 2 Unfolded Pipelined CORDIC Architecture

The CORDIC algorithm has been implemented conventionally using bit serial architecture with all iterations are executed in the same hardware. This slowed down the computational device and therefore, is not suitable for high speed implementation. The word serial architecture is an iterative CORDIC architecture obtained by realizing the iteration equations. In this architecture, the shifters are modified in each iteration to cause the desired shift for the iteration. The appropriate elementary angles σ_i are accessed from a lookup table. The most hindering speed factors during the iterations of word serial architecture are carry/borrow propagate addition/subtraction and variable shifting operations, rendering the conventional CORDIC implementation slow for high speed applications. These disadvantages were overcome by using unfolded architectures. The main advantage of the unfolded pipelined architecture compared to folded architecture is high throughput due to the hardwired fixed position shifters rather than the time and area consuming barrel shifters and the removal of ROM. Pipelined architectures offer a throughput improvement by a factor of n for n-bit precision at the expense of increasing the hardware by a factor less than n [5].

Several parallel CORDIC algorithms where stated in the literature. These algorithms try to increase the speed of the CORDIC by obtaining σ_i values and/or x/y coordinatesin parallel instead of sequentially. Examples include LowLatency Radix-2 CORDIC [9], Hybrid CORDIC [15], [18], Flat CORDIC [24], P-CORDIC [21], Para-CORDIC [22], Semi-flat Cordic [23], [26], [31], [33]. The scaling factor for most of these algorithms is constant as σ_i is restricted totake values form the set {-1, 1}. The main disadvantage of using some of these algorithms is that they require additional complex hardware and have poor scalability [5].

IV. Multi Cordic Algorithm

Similar to [15], [17], the multi CORDIC algorithm consists of three stages: argument reduction, CORDIC calculation and output normalization. The argument reduction stage transforms the inputs from floating-point format into fixed-point format and expands the convergence range of the CORDIC algorithm. CORDIC calculation stage evaluates the desired CORDIC function based on the selected CORDIC architecture. Normalization stage transforms the outputs back to floating-point format and standardizes them into IEEE STD 754-1985 format.

A. Argument Reduction Algorithm

The argument reduction algorithm was done using in argument reduction phase which is based on [15] and [17]. IEEE double-precision Floating-Point was used as the format for both the input and output data. To clarify further suppose the following notations:

(1) Floating-Point Format of the Input Data:

$$X = Sx * 2^{Ex} * Mx,\ Y = Sy * 2^{Ey} * My,\ Z = Sz * 2^{Ez} * Mz, = Sz * Mz'',\ Sx, Sy, Sz \epsilon \{-1,1\}$$

(2) Given Eref=max (Ex, Ey), Mx'' and My'' are fixed-point formats of X and Y expressed as:

$$Mx'' = Mx * 2^{Ex-Eref},\ My'' = My * 2^{Ey-Eref}$$

1. m = 0 (Linear Coordinate):

Floating-point multiplication and division operation are implemented in this coordinate, in which no real argument reduction is required. However, there are some adjustments necessary which are summarized in the following table:

TABLE II
Summary of Operations for M = 0

	Rotation	Vectoring
Input	$Y_0 = 0$	$Z_0 = 0$
Step 1	$Eref = Ex + Ez$	$Eref = Ey - Ex$
Step 2	$My_n = Mx * Mz$	$Mz_n = My/Mx$
Step 3	$Y_n = Sx * Sz * Norm(My_n * 2^{Eref})$	$Z_n = Sx * Sy * Norm(Mz_n * 2^{Eref})$

2. m = 1 (Circular Coordinate):

a) Rotation:

The input angle Z is mapped into the domain of [0, π/2] using the property of periodicity:

$$Z = Mz'' = \frac{\pi}{2} * Q + D = \frac{\pi}{2} * (Q + R) \text{ (Given)}$$

$$Q + R = \frac{2}{\pi} * Mz'' \quad (2)$$

$$Mz' = D = \frac{\pi}{2} * R \quad (3)$$

where Q is an integer denoting the quadrant of angle Z, R is the fraction and D is the mapped angle in the domain of [0,π/2].

TABLE III
Quadrant Mapping

Domain	Q[1:0]	Mx'	My'
[0,π/2]	00	Mx''	My''
[π/2,π]	01	$-My''$	Mx''
[π,3 π/2]	10	Mx''	$-My''$
[3π/2,2π]	11	My''	$-Mx''$

$Mx', My'$are chosen from Table III depending on the two least significant bits of Q. Afterwards, Mx', My' *and* Mz' are taken as inputs to the CORDIC calculation phase and Mx_n, My_n are results. Then cos(Z) = $2^{Eref} Mx_n$ and sin(Z) = $2^{Eref} My_n$ holds.

b) Vectoring:

Mx'', My'' *and* Mz'' are directly regarded as the inputs of the CORDIC calculation phase and Mx_n, Mz_n are results. Then $\sqrt{X^2 + Y^2} = 2^{Eref} Mx_n$ holds. For the calculation of arctangent ($tan^{-1}(Y/X)$) fix is still needed on the quadrant of Mz_n according to the sign of X and Y as follows:

$$Mz_n = \begin{cases} Sy * Mz_n & Sx = 1 \\ Sy * (\pi - Mz_n) & Sx = -1 \end{cases} \quad (4)$$

3. m = -1 (Hyperbolic Coordinate)

a) Rotation:

The input angle Z is mapped into the domain of [0, ln(2)] using the property of periodicity:

$$Z = Mz'' = \ln(2) * Q + D = \ln(2) * (Q + R) \text{ (Given)}$$

$$Q + R = \frac{1}{\ln(2)} * Mz'' \quad (5)$$

$$Mz' = D = \ln(2) * R \quad (6)$$

where Q is an integer denoting the quadrant of angle Z, R is the fraction and D is the mapped angle in the domain of [0, ln(2)]. Given:

$$\begin{cases} Mx' = (Mx'' + My'') + 2^{-2Q}(Mx'' - My'') \\ My' = (Mx'' + My'') - 2^{-2Q}(Mx'' - My'') \end{cases} \quad (7)$$

Mx', My' *and* Mz' are taken as inputs to the CORDIC calculation phase and Mx_n, My_n are results. Then cosh(Z) = $2^{Eref+Q-1} Mx_n$ and sinh(Z) = $2^{Eref+Q-1} My_n$ holds.

b) Vectoring:

If $Mx'' \geq 2My''$ then Mx'', My'' and Mz'' are taken as inputs tothe CORDIC calculation phase and $Mx_n, Mz_n$are results. Then $\sqrt{X^2 - Y^2} = 2^{Eref} Mx_n$ holds. For the calculation of hyperbolic arctangent ($\tanh^{-1}(Y/X)$) the sign of $\mathrm{Mz_n}$ needs to be properly assigned based to the sign of X and Y as follows:

$$\mathrm{Mz_n} = \mathrm{Sy} * \mathrm{Sx} * \mathrm{Mz_n}$$

For $Mx'' < 2My''$, let γ represent the number of leading zeroes in ($\mathrm{Mx''} - \mathrm{My''}$). Given:

$$\mathrm{Enew} = \begin{cases} 1, & \gamma = 1 \\ \gamma - 1, & \gamma > 1 \end{cases} \quad (8)$$

Then

$$\begin{cases} \mathrm{Mx'} = (\mathrm{Mx''} + \mathrm{My''}) + 2^{-\mathrm{Enew}}(\mathrm{Mx''} - \mathrm{My''}) \\ \mathrm{My'} = (\mathrm{Mx''} + \mathrm{My''}) - 2^{-\mathrm{Enew}}(\mathrm{Mx''} - \mathrm{My''}) \end{cases} \quad (9)$$

$Mx', My' \text{ and } Mz'$ are taken as inputs to the CORDIC calculation phase and$\mathrm{Mx_n}, \mathrm{Mz_n}$ are results. Then

$$\tan^{-1}(\mathrm{Y/X}) = \mathrm{Sy} * \mathrm{Sx} * (\mathrm{Mz_n} + 0.5 * \mathrm{Enew} * \ln(2))$$

holds.

For calculation of square-root ($\sqrt{X^2 - Y^2}$)Mx_n needs to be scaled based on Enew as follows:

$$Mx_n = \begin{cases} Mx_n * 2^{\mathrm{Eref} - (\frac{\mathrm{Enew}+2}{2})}, & \text{Enew mode 2} = 0 \\ (\frac{1}{\sqrt{2}}) Mx_n * 2^{\mathrm{Eref} - (\frac{\mathrm{Enew}+1}{2})}, & \text{Enew mode 2} = 1 \end{cases} \quad (10)$$

In addition, the following functions are derived from the previous functions [16]:

$\ln(\mathrm{W}) = 2 * \tan^{-1}(\mathrm{Y/X})$ where X = W + 1 and Y = W − 1.

$\sqrt{W} = \sqrt{X^2 - Y^2}$ where X = W + (1/4) and Y = W – (1/4).

V. Multi Cordic Processor Implementation

A pipelined 64-bit IEEE floating-point multi CORDIC processor was implemented on FPGA platform. The purpose was to implement different CORDIC architectures on the same FPGA chip were the user can choose which CORDIC architecture to use for the calculation according to a selection function. As each CORDIC architecture can only calculate specific functions, the choices will be limited by what the user wants to calculate.

The multi CORDIC architecture was implemented in Verilog, synthesized with Xilinx vertex 7 FPGA XC7VX980T to ensure enough area for mapping the architecture (Xilinx vertex 5 and 6 did not have enough area) and includes the following CORDIC architectures: radix-2 CORDIC, radix-2 hybrid-mode CORDIC [15], radix-4 CORDIC [12], [14], [19], [20], radix-4 hybrid-mode CORDIC, Para-CORDIC [22] and radix-8 CORDIC. All can run in both rotation and vectoring modes except Para-CORDIC and radix-8 CORDIC which run in rotation mode only. Also radix-4 CORDIC, radix-4 hybrid-mode CORDIC and radix-8 CORDIC currently cannot run in hyperbolic coordinate systems.

The multi CORDIC architecture is implemented as follows: Similar to [15], the multi CORDIC architecture contains a pre-process module and a post-process module in addition to the CORDIC modules (for each separate CORDIC architecture). The inputs to the multi CORDIC architecture include the x,y and z inputs, the x,y and z outputs, the coordinate selection mode m, the vectoring mode selection vm and the clock and rest signals in addition to the CORDIC selection function (which selects which CORDIC architecture to proceed with). Before the pre-process module, the input to the selected CORDIC architecture is set while reset is set to 1 for the other CORDIC architectures. This is done as the other CORDIC architectures have no use. The inputs to the post-process module are also directly set based on the selected CORDIC architecture after the CORDIC modules. More specifically the pipeline stages of the multi CORDIC architecture are as follows:

S1: Setting the rst input of all CORDIC architectures

The rst (rest) input of the selected CORDIC architecture is set to the rst input of the multi CORDIC architecture while all other CORDIC architectures have their rst input set to 1.
x, y and z inputs propagated.

S2: Converting the floating-point format of inputs into fixed point format.

Mantissas: 1 is added to the left of the mantissas of X, Y and Z as it is excluded when mantissas are not equal to0. Otherwise if a mantissa = 0 no 1 is added to the left of it. Next, given Eref3 = max(Ex, Ey), (Ex − Eref3) bits right shift of the mantissa of X produces Mx" and (Ey − Eref3)bits right shift of the mantissa of Y produces My". IfEz> 1023, (Ez − 1023) bits left shift of the mantissa Z produces Mz" or else (1023 − Ez) bits right shift of the mantissa Z produces Mz". For the calculation of natural logarithm: Mx"= Mx" + (1 shifted left by (52 − Eref3 − 1023) bits) and My" = My" − (1 shifted left by (52 − Eref3 − 1023) bits).

TABLE IV
COMPARISONS OF THE SYNTHESIS RESULTS OF THE DIFFERENT CORDIC ARCHITECTURES ON FPGA

CORDIC Architecture	Path Delay	Number of Slice LUTs	Number of Slice Registers	Power
Radix 2 CORDIC	8.398ns	24473 out of 612000 (3%)	11,819 out of 1224000 (1%)	0.331W
Hybrid Radix 2 CORDIC	8.497ns	22758 out of 612000 (3%)	10981 out of 1224000 (1%)	0.331W
Radix 4 CORDIC	13.977ns	21743 out of 612000 (3%)	7858 out of 1224000 (1%)	0.331W
Hybrid Radix 4 Cordic	14.232ns	20717 out of 612000 (3%)	6952 out of 1224000 (1%)	0.331W
Para-CORDIC	9.346ns	23353 out of 612000 (4%)	11020 out of 1224000 (1%)	0.331W
Radix 8 CORDIC	12.477ns	37254 out of 612000 (6%)	6299 out of 1224000 (0.5%)	0.331W
Multi CORDIC	19.544ns	100069 out of 612000 (16%)	36803 out of 1224000 (3%)	0.331W

For the calculation of square root:

Mx" = Mx" + (1 shifted left by (50 − Eref3 − 1023) and My" = My" − (1 shifted left by (50 − Eref3 − 1023).

Exponents: For the calculation of multiplication, Eref1 = Ex + Ez − 1023 and for the calculation of division, Eref2 = Ey − Ex +1023. Otherwise, Eref3 = max(Ex, Ey).

S3 - S4: Scaling Factor Compensation and Angle mapping in argument reduction

Mantissas: X and Y need to be scaled with the constant scaling factor Km depending on the CORDIC selection. Scaling factors are gotten in S2 while in S3, X and Y are compensated with the corresponding scaling factor depending on the CORDIC selection. For rotation mode, mapping the angle Z into the destined domain needs a fixed-point constant multiplication. For circular coordinate, a constant multiplier is required to execute (2). For hyperbolic coordinate, a constant multiplier is required to execute (5).

In S2, the value of Mx" − is My" is gotten which is used in S3 to get the value of γ through the module of LeadZero. Then, the magnitude of Enew is gotten through (8).

Exponents and signs Propagated.

S5: Calculation mapped angle value in argument reduction.

Mantissas: selecting Mx' and My' according to Table III and the magnitude of Q. Accomplishing the computation of (7) and (9) by two adders, one subtractor and two shifters. Two constant multipliers are employed to execute the computation of (3) and (6) respectively.

Exponents and signs Propagated

S6: Selecting the inputs of the CORDIC calculation module according to the coordinate mode m, the operation mode vm and the corresponding function (to be evaluated) fn.

S7-SN: CORDIC calculation phase.

Each CORDIC architecture is allowed to run on its own. Each stage accomplishes one or more iterations of the CORDIC depending the CORDIC architecture selected. N is used here to indicate that the number of stages is variable depending on the selected CORDIC architecture.

SN+1: Setting the inputs of the PostProcess module.

Mantissas: For natural exponential ex, if Sz equals 0 then X and Y, are added to product the final value of ex else ex = X - Y. For natural logarithm ln(x) or hyperbolic arctangent arctanh(x), Z and Exln2, which are the output of CORDIC process module and Mult ln2 module respectively, are added to product the final value of ln(x) or arctanh(x). For arctangent arctan(x), accomplishing (4) products the final value of arctan(x). If X55 < 0 then the final value X = -X. If Y <0 then the final value Y = -Y.

Signs: For final value of X, Sx = Sign(X). For final value of Y, Sy = xor(Sx,Sz) if(mode m = 0) else Sy = xor(Sign(Y),Sz). For Final value of Z, Sz = Syif(mode m = 1) elseSz = xor(Sx,Sy).

Exponents propagated.

SN+3: Counting leading zeros

Mantissas: Counting leading zeros in the final values of X, Y, Z and ex.

Exponents propagated.

SN+4: Normalization of output.

X, Y, Z, and ex (not including the sign bit) are shifted to the left according to the number of the leading zeroes and are then truncated to form the mantissas of the outputs respectively. The first 1 bit (not including the sign bit) is not included as in the IEEE 754 floating-point format.

For X, Y and ex, the exponent of the output is calculated by subtracting the number of the leading zeroes of X, Y and ex from Eref respectively. If the length of X, Y or ex (after shifting and not including the first 1 bit) is greater than 52bits (before truncation) then the length (of X or Y or Z) – 52must be added to each exponent output. The exponent of Z is computed by subtracting the number of the leading zeros ofZ from Eref if(mode m = 0 and operation mode vm = 1) or from 1023 otherwise. As with other exponents, the length ofZ - 52 must be added to the exponent of Z if length of Z is> 52. +1 is added to the Z exponent for the calculation of the natural logarithm. Comparison of the synthesis results of the multi-CORDIC architecture with respect to the other CORDIC architecture simplemented on FPGA (Xilinx vertex 7 FPGA XC7VX980T) are summarized in Table IV.

VI. EXPERIMENTS

Experiments were conducted in order to observe the accuracy and precision of functions computed by the different CORDIC architectures. The results are compared with results produced by a Pentium processor and are shown on Table V where the relative error is shown on the right most column of each sub-table. The value shown in the relative error column is a value i such that $2^{-i} \geq$ relative error $> 2^{-i+1}$. Many ofthe results show that the designs were correct. Relative errorgenerally reaches a maximum error of 2^{-27} for small data anda maximum error of 2^{-39} large data.

VII. Conclusion

This paper presented a multi CORDIC architecture which was implemented on FPGA and it included different CORDIC architectures combined where the user can select the cordic architecture that suits his needs. It was organized into three phases, Argument reduction, CORDIC calculations and output normalization. Synthesis show that radix 2 CORDIC has clock delay, radix 8 CORDIC has the highest LUT usage and lowest register usage while Hybrid Radix 4CORDIC had the highest clock delay. Error results show that normally a maximum error of $2^{-27}$is reached for small data and a maximum error of 2^{-39} is reached for large data.Generally, there is no clear winner for which architectureachieves the better precision but there are special cases whereone or more architectures dominate. For example, Radix 4 CORDIC and Hybrid Radix 4 CORDIC achieve the best precision for the calculation of Arctan while Para CORDIC achieves the best precision for the calculations of sinh, cosh and exp. So, the multi CORDIC processor can be used as a component in scientific computations.

TABLE V
Experimental Results (a) Accuracy and Precision Test of Sin

Operand	CORDIC Architecture	Pentium Result	CORDIC Result	$E(2^{-i})$
0.000001	CORDIC Radix-2	9.999999999998333E-7	9.999999941445736E-7	27
0.000001	CORDIC Radix-2 Hybrid-mode	9.999999999998333E-7	9.999999941445736E-7	27
0.000001	CORDIC Radix-4	9.999999999998333E-7	9.999999994736442E-7	50
0.000001	CORDIC Radix-4 Hybrid-mode	9.999999999998333E-7	9.999999992515995E-7	31
0.000001	Para CORDIC	9.999999999998333E-7	9.999999997858177E-7	32
0.000001	CORDIC Radix-8	9.999999999998333E-7	9.999999992515995E-7	30
π / 6	CORDIC Radix-2	0.49999999999999994	0.49999999999999334	46
π / 6	CORDIC Radix-2 Hybrid-mode	0.49999999999999994	0.49999999999999334	46
π / 6	CORDIC Radix-4	0.49999999999999994	0.5000000000002425	41
π / 6	CORDIC Radix-4 Hybrid-mode	0.49999999999999994	0.5000000000002425	41
π / 6	Para CORDIC	0.49999999999999994	0.49999999999999983	52
π / 6	CORDIC Radix-8	0.49999999999999994	0.5000000000002294	41
1024.0	CORDIC Radix-2	-0.15853338004399595	-0.15853338004393946	41
1024.0	CORDIC Radix-2 Hybrid-mode	-0.15853338004399595	-0.15853338004393946	41
1024.0	CORDIC Radix-4	-0.15853338004399595	-0.15853338004424167	39
1024.0	CORDIC Radix-4 Hybrid-mode	-0.15853338004399595	-0.15853338004424167	39
1024.0	Para CORDIC	-0.15853338004399595	-0.15853338004393386	41
1024.0	CORDIC Radix-8	-0.15853338004399595	-0.15853338004409712	41

(b) Accuracy and Precision Test of Cos

Operand	CORDIC Architecture	Pentium Result	CORDIC Result	$E(2^{-i})$
0.000001	CORDIC Radix-2	0.9999999999995	0.9999999999995064	47
0.000001	CORDIC Radix-2 Hybrid-mode	0.9999999999995	0.9999999999995064	47
0.000001	CORDIC Radix-4	0.9999999999995	0.9999999999999578	41
0.000001	CORDIC Radix-4 Hybrid-mode	0.9999999999995	0.999999999999958	41
0.000001	Para CORDIC	0.9999999999995	1.0000000000067764	37
0.000001	CORDIC Radix-8	0.9999999999995	0.999999999999958	41
π / 6	CORDIC Radix-2	0.8660254037844387	0.8660254037844466	46
π / 6	CORDIC Radix-2 Hybrid-mode	0.8660254037844387	0.8660254037844466	46
π / 6	CORDIC Radix-4	0.8660254037844387	0.8660254037848636	41
π / 6	CORDIC Radix-4 Hybrid-mode	0.8660254037844387	0.8660254037848636	41
π / 6	Para CORDIC	0.8660254037844387	0.8660254037844386	53
π / 6	CORDIC Radix-8	0.8660254037844387	0.8660254037848392	41
1024.0	CORDIC Radix-2	0.9873536182198484	0.9873536182198626	46
1024.0	CORDIC Radix-2 Hybrid-mode	0.9873536182198484	0.9873536182198626	46
1024.0	CORDIC Radix-4	0.9873536182198484	0.9873536182217675	39
1024.0	CORDIC Radix-4 Hybrid-mode	0.9873536182198484	0.9873536182217675	39
1024.0	Para CORDIC	0.9873536182198484	0.9873536182270427	37
1024.0	CORDIC Radix-8	0.9873536182198484	0.9873536182208698	40

(C) ACCURACY AND PRECISION TEST OF ARCTAN

Operand	CORDIC Architecture	Pentium Result	CORDIC Result	$E(2^{-i})$
tan(0.0001)	CORDIC Radix-2	0.0001	9.999999999375352E-5	34
tan(0.0001)	CORDIC Radix-2 Hybrid-mode	0.0001	9.999999999375135E-5	34
tan(0.0001)	CORDIC Radix-4	0.0001	9.999999999873131E-5	36
tan(0.0001)	CORDIC Radix-4 Hybrid-mode	0.0001	9.999999999873175E-5	36
2.0	CORDIC Radix-2	1.1071487177940904	1.1071487177940855	48
2.0	CORDIC Radix-2 Hybrid-mode	1.1071487177940904	1.1071487177940855	48
2.0	CORDIC Radix-4	1.1071487177940904	1.1071487177940902	52
2.0	CORDIC Radix-4 Hybrid-mode	1.1071487177940904	1.1071487177940902	52
7968578.0	Cordic Radix-2	1.5707962013019918	1.570796201301989	49
7968578.0	CORDIC Radix-2 Hybrid-mode	1.5707962013019918	1.570796201301989	49
7968578.0	CORDIC Radix-4	1.5707962013019907	1.570796201301989	49
7968578.0	CORDIC Radix-4 Hybrid-mode	1.5707962013019907	1.570796201301989	49

(D) ACCURACY AND PRECISION TEST OF SINH

Operand	CORDIC Architecture	Pentium Result	CORDIC Result	$E(2^{-i})$
0.000001	CORDIC Radix-2	1.0000000000001666E-6	9.999999974752427E-7	29
0.000001	CORDIC Radix-2 Hybrid-mode	1.0000000000001666E-6	9.99999997586265E-7	29
0.000001	Para CORDIC	1.0000000000001666E-6	1.0000000001717992E-6	32
5.65	CORDIC Radix-2	142.143974153573	142.1439741535723	48
5.65	CORDIC Radix-2 Hybrid-mode	142.143974153573	142.14397415357234	48
5.65	Para CORDIC	142.143974153573	142.14397415357288	50
100	CORDIC Radix-2	1.3440585709080678E43	1.344058570908060E43	47
100	CORDIC Radix-2 Hybrid-mode	1.3440585709080678E43	1.344058570908060E43	47
100	Para CORDIC	1.3440585709080678E43	1.3440585709080802E43	47

(E) ACCURACY AND PRECISION TEST OF COSH

Operand	CORDIC Architecture	Pentium Result	CORDIC Result	$E(2^{-i})$
0.000001	CORDIC Radix-2	1.0000000000005	1.0000000000004976	49
0.000001	CORDIC Radix-2 Hybrid-mode	1.0000000000005	1.0000000000004976	49
0.000001	Para CORDIC	1.0000000000005	1.000000000000499	50
5.65	CORDIC Radix-2	142.14749167034793	142.14749167034716	47
5.65	CORDIC Radix-2 Hybrid-mode	142.14749167034793	142.14749167034716	47
5.65	Para CORDIC	142.14749167034793	142.1474916703479	53
100	CORDIC Radix-2	1.3440585709080678E43	1.344058570908060E43	47
100	CORDIC Radix-2 Hybrid-mode	1.3440585709080678E43	1.344058570908060E43	47
100	Para CORDIC	1.3440585709080678E43	1.3440585709080802E43	47

(F) ACCURACY AND PRECISION TEST OF EXP

Operand	CORDIC Architecture	Pentium Result	CORDIC Result	$E(2^{-i})$
0.000001	CORDIC Radix-2	1.0000010000005	1.000001000000495	48
0.000001	CORDIC Radix-2 Hybrid-mode	1.0000010000005	1.0000010000004953	48
0.000001	Para CORDIC	1.0000010000005	1.000001000000499	50
5.65	CORDIC Radix-2	284.29146582392093	284.29146582391945	47
5.65	CORDIC Radix-2 Hybrid-mode	284.29146582392093	284.29146582391957	48
5.65	Para CORDIC	284.29146582392093	284.2914658239208	51
100	CORDIC Radix-2	2.6881171418161356E43	2.688117141816121E43	47
100	CORDIC Radix-2 Hybrid-mode	2.6881171418161356E43	2.688117141816121E43	47
100	Para CORDIC	2.6881171418161356E43	2.6881171418161604E43	47

(G) ACCURACY AND PRECISION TEST OF ARCTANH

Operand	CORDIC Architecture	Pentium Result	CORDIC Result	$E(2^{-i})$
0.0001	CORDIC Radix-2	1.0000000033336145E-4	1.0000000032587154E-4	34
0.0001	CORDIC Radix-2 Hybrid-mode	1.0000000032942121E-4	1.0000000032587154E-4	35
0.5	CORDIC Radix-2	0.5493061443340549	0.5493061443340521	47
0.5	CORDIC Radix-2 Hybrid-mode	0.5493061443340549	0.5493061443340522	48
0.99975	CORDIC Radix-2	4.493535906424466	4.493535906424107	44
0.99975	CORDIC Radix-2 Hybrid-mode	4.493535906424466	4.493535906424107	44

(H) ACCURACY AND PRECISION TEST OF LN

Operand	CORDIC Architecture	Pentium Result	CORDIC Result	$E(2^{-i})$
0.001	CORDIC Radix-2	-6.907755278982137	-6.907755278982137	∞
0.001	CORDIC Radix-2 Hybrid-mode	-6.907755278982137	-6.907755278982136	53
4.0	CORDIC Radix-2	1.3862943611198906	1.3862943611198868	48
4.0	CORDIC Radix-2 Hybrid-mode	1.3862943611198906	1.3862943611198866	48
1.0E13	CORDIC Radix-2	29.933606208922594	29.933606208922594	∞
1.0E13	CORDIC Radix-2 Hybrid-mode	29.933606208922594	29.933606208922594	53

(I) ACCURACY AND PRECISION TEST OF SQRT

Operand	CORDIC Architecture	Pentium Result	CORDIC Result	$E(2^{-i})$
0.001	CORDIC Radix-2	0.03162277660168379	0.0316227766016838	52
0.001	CORDIC Radix-2 Hybrid-mode	0.03162277660168379	0.0316227766016838	52
4.0	CORDIC Radix-2	2.0	1.999999999999992	48
4.0	CORDIC Radix-2 Hybrid-mode	2.0	1.9999999999999922	48
1.0E9	CORDIC Radix-2	31622.776601683795	31622.7766016838	52
1.0E9	CORDIC Radix-2 Hybrid-mode	31622.776601683795	31622.7766016838	52

REFERENCES

[1] J. E. Volder, "The cordic trigonometric computing technique," Electronic Computers, IRE Transactions on, vol. EC-8, no. 3, pp. 330–334, 1959.

[2] J. E. Volder, "The birth of cordic," J. VLSI Signal Process. Syst., vol. 25, no. 2, pp. 101–105, Jun. 2000. [Online]. Available: http://dx.doi.org/10.1023/A:1008110704586

[3] J. S. Walther, "A unified algorithm for elementary functions," in Proceedings of the May 18-20, 1971, spring joint computer conference, ser. AFIPS '71 (Spring). New York, NY, USA: ACM, 1971, pp. 379 – 385.

[4] J. S. Walther, "The story of unified cordic," J. VLSI Signal Process. Syst., vol. 25, no. 2, pp. 107–112, Jun. 2000. [Online]. Available: http://dx.doi.org/10.1023/A:1008162721424

[5] B. Lakshmi and A. S. Dhar, "Cordic architectures: a survey," VLSI Des., vol. 2010, pp. 2:1–2:7, Jan. 2010.

[6] E. Antelo, J. Villalba, and E. L. Zapata, "A low-latency pipelined 2d and 3d cordic processors," Computers, IEEE Transactions on, vol. 57, no. 3, pp. 404–417, 2008.

[7] M. Ercegovac and T. Lang, "Fast cosine/sine implementation using on-line coric," in Proceedings of the 21st Asilomar Conference on Signals, Systems, and Computers, 1987.

[8] N. Takagi, T. Asada, and S. Yajima, "Redundant cordic methods with a constant scale factor for sine and cosine computation," IEEE Trans. Comput., vol. 40, no. 9, pp. 989–995, Sep. 1991.

[9] D. Timmermann, H. Hahn, and B. J. Hosticka, "Low latency time cordic algorithms," IEEE Trans. Comput., vol. 41, no. 8, pp. 1010–1015, Aug. 1992.

[10] J. Duprat and J. M. Muller, "The cordic algorithm: New results for fast vlsi implementation," IEEE Trans. Comput., vol. 42, no. 2, pp. 168–178, Feb. 1993.

[11] H. Dawid and H. Meyr, "The differential cordic algorithm: Constant scale factor redundant implementation without correcting iterations," IEEE Trans. Comput., vol. 45, no. 3, pp. 307–318, Mar. 1996. [Online], Available: http://dx.doi.org/10.1109/12.485569

[12] J. D. Bruguera, E. Antelo, and E. L. Zapata, "Design of a pipelined radix 4 cordic processor," Parallel computing, vol. 19, no. 7, pp. 729–744,1993.

[13] E. Antelo, J. D. Bruguera, and E. L. Zapata, "Unified mixed radix 2-4 redundant cordic processor," Computers, IEEE Transactions on, vol. 45, no. 9, pp. 1068–1073, 1996.

[14] E. Antelo, J. Villalba, J. D. Bruguera, and E. L. Zapata, "High performance rotation architectures based on the radix-4 cordic algorithm," Computers, IEEE Transactions on, vol. 46, no. 8, pp. 855–870, 1997.

[15] L. Y. X. J. . D. Y. "Zhou J., Dou Y., "double precision hybrid-mode floating-point fpga cordic co-processor," in "2008 10th IEEE International Conference on High Performance Computing and Communications", pp. 182 – 198.

[16] R. Andraka, "A survey of cordic algorithms for fpga based computers," in Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays, ser. FPGA '98. New York, NY, USA: ACM, 1998, pp. 191–200.

[17] Hahn, D. Timmermann, B. J. Hosticka, and B. Rix, "A unified and division-free cordic argument reduction method with unlimited convergence domain including inverse hyperbolic functions," IEEE Trans. Comput., vol. 43, no. 11, pp. 1339–1344, Nov. 1994.

[18] P. S. Wang and E. W. Jr., "Hybrid cordic algorithms," IEEE Transactions on Computers, vol. 46, no. 11, pp. 1202–1207, 1997.

[19] K. Bhattacharyya, R. Biswas, A. S. Dhar, and S. Banerjee, "Architectural design and fpga implementation of radix-4 cordic processor," Microprocess. Microsyst., vol. 34, no. 2-4, pp. 96–101, Mar. 2010.

[20] J. Villalba, E. L. Zapata, E. Antelo, and J. D. Bruguera, "Radix-4 vectoring cordic algorithm and architectures," J. VLSI Signal Process. Syst., vol. 19, no. 2, pp. 127–147, Jul. 1998.

[21] M. Kuhlmann and K. K. Parhi, "P-cordic: a precomputation based rotation cordic algorithm," EURASIP J. Appl. Signal Process., vol. 2002, no. 1, pp. 936–943, Jan. 2002.

[22] T.-B. Juang, S.-F. Hsiao, and M.-Y. Tsai, "Para-cordic: parallel cordic rotation algorithm," Circuits and Systems I: Regular Papers, IEEE Transactions on, vol. 51, no. 8, pp. 1515 – 1524, aug. 2004.

[23] H. Kebbati, J. Blonde, and F. Braun, "A new semi-flat architecture for high speed and reduced area cordic chip," Microelectronics Journal, vol. 37, no. 2, pp. 181 – 187, 2006.

[24] B. Gisuthan and T. Srikanthan, "Pipelining flat cordic based trigonometric function generators," Microelectronics Journal, vol. 33, pp. 77 – 89, 2002.

[25] H. N.-u.-d. BurhanKhurshid, 2Gulam Mohd Rather, "Performance comparison of non-redundant and redundant fpga based unfolded cordic architectures," IJECT, vol. 3, no. 1, pp. 85–89, Jan. 2012.

[26] D.-M. Ross, S. Miller, M. Sima, and M. McGuire, "Exploration of sign precomputation-based cordic in reconfigurable systems," in Signals, Systems and Computers (ASILOMAR), 2011 Conference Record of the Forty Fifth Asilomar Conference on. IEEE, 2011, pp. 2186–2191.

[27] J. Valls, M. Kuhlmann, and K. K. Parhi, "Evaluation of cordic algorithms for fpga design," Journal of VLSI signal processing systems for signal, image and video technology, vol. 32, no. 3, pp. 207–222, 2002.

[28] S. Vadlamani and W. Mahmoud, "Comparison of cordic algorithm implementations on fpga families," in System Theory, 2002. Proceedings of the Thirty-Fourth Southeastern Symposium on. IEEE, 2002, pp. 192–196.

[29] D. Yi, "Cordic algorithm based on fpga," J Shanghai Univ (Engl Ed), vol. 15, no. 4, pp. 304–309, 2011.

[30] R. Bhakthavatchalu, M. Sinith, P. Nair, and K. Jismi, "A comparison of pipelined parallel and iterative cordic design on fpga," in Industrial and Information Systems (ICIIS), 2010 International Conference on. IEEE, 2010, pp. 239–243.

[31] Y. Chandrakanth and M. Kumar, "Low latency &high precision cordic architecture using improved parallel angle recoding," in Signal Processing, Communication, Computing and Networking Technologies (ICSCCN), 2011 International Conference on. 498–501.

[32] J. Rudagi, B. B. Srikant, and S. Subbaraman, "Performance analysisof radix 4 cordic processor in rotation mode with parallel scalefactor computation," International Journal of Emerging Technology and Advanced Engineering, vol. 2, no. 7, pp. 507–510, 2012.
[33] D. Timmermann and I. Sundsbo, "Area and latency efficient cordic architectures," in Circuits and Systems, 1992. ISCAS '92. Proceedings, 1992 IEEE International Symposium on, vol. 3, 1992, pp. 1093–1096.