Complex Division and Square-Root Using CORDIC

Bohan Yang, Dong Wang and Leibo Liu
Institute of Microelectronics
Tsinghua University
Beijing 100084 P.R.C.

***Abstract*—In this paper, we present designs of CORDIC-based fixed-point complex division and square-root FPGA implementations. We optimized the arithmetic of CORDIC unit, complex division and square-root. Then a architecture of CORDIC units based recurrence architecture was proposed. The complex number division and square-root was implement and evaluated on a Altera Stratix II FPGAs, with max frequency up to 180Mhz, and can easily be ported to ASIC implementation. These architecture have good numerical properties and the result is accurate to less than one ulp.**

***Keywords*—CORDIC, Complex, Division, Square-root, FPGA.**

I. Introduction

COMPLEX operations are a necessity and widely used in modern science [1] and industry [2]. Especially, due to the rapid development of personal mobile devices, the complex numerical arithmetic is used in many aspect of these wireless communication systems [3]. There is a growing needs of numerical accuracy, area economized and calculation speeded hardware architectures. In responses to this problem, fixed-point complex number arithmetic units are proposed in this paper.

Currently, conventional commercial DSP and embedded processors do not provided specific complex number computation unit. Instead of dedicated arithmetic units, they often use a serial of combined real elementary operations to compute complex number in software level. The redundancy from control operation of decomposed solution results in decrease performance of computations.

In this paper, we proposed an architecture for complex functions and present its hardware implementation. The implementation was based on CORDIC algorithm. The CORDIC unit only consists of two kinds of arithmetic units, adder and shifter. In the beginning of this paper, we optimized the algorithm for hardware implementation. Then we proposed architectures for complex valued divider and square-root.

II. Review of the CORDIC Algorithm

A. Notations

Throughout this paper, we discuss the algorithm and implementation of arithmetic unit for complex numbers in fixed-point format. We shall use the following notation:

This work is supported by National Natural Science Foundation of China(NSFC) No.61106022

- i represents $\sqrt{-1}$;
- Complex number is denoted as $z = a + ib$;
- $\Re(z)$ and $\Im(z)$ denote the real and imaginary parts.
- K and K' are modifications for CORDIC algorithm.
- norm $\|Z\|_\infty$ denotes $Max\{|\Re(Z)|, |\Im(Z)|\}$.

B. The general CORDIC algorithm

CORDIC is the acronym of a trigonometric algorithm for COordinate Rotation Digital Computer. It has been first introduced by Jack Volder [4] and later extended investigation by Walther [5]. CORDIC is an iteration algorithm based on vector rotations and it only consists of shifts and add. However, it can generate solutions for trigonometric and some transcendental functions.

This algorithm is derived from the general (Givens) rotation transform:

$$\begin{aligned}\begin{bmatrix} X' \\ Y' \end{bmatrix} &= \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix} \times \begin{bmatrix} X \\ Y \end{bmatrix} \\ &= \cos\Theta \begin{bmatrix} 1 & -\tan\Theta \\ \tan\Theta & 1 \end{bmatrix} \times \begin{bmatrix} X \\ Y \end{bmatrix}\end{aligned}$$

Some modifications have taken to simplify and generalize the CORDIC arithmetic. Firstly, a series of fixed angles are linear combined to approximate the rotated angle. Then the real rotation is replaced with pseudo-rotation to reduce computational complexity for each iteration step. The variable Z allows us to keep track of angles reminded to be rotated. Due to the pseudo-rotation, the final vector length increased by: $K = \prod_{i=0}^{\infty}(1+\tan^2\theta_i)^{\frac{1}{2}}$(for circular mode). CORDIC method can be generalized by introducing a parameter μ and redefining the dedicated angle serials. Through these modifications, algorithm is extend to perform more functions. Generalized CORDIC iteration is shown as below equations. TABLE I generalized six different modes and their functions, K and K' shown in this table are the modification for Circular and hyperbolic modes, the values are $K = 1.646760258121 \quad K' = 0.8281593609602$.

$$\begin{aligned} X_{i+1} &= X_i - \mu d_i Y(i) 2^{-i} \\ Y_{i+1} &= Y_i + d_i X(i) 2^{-i} \\ Z_{i+1} &= Zi - d_i \theta_i \end{aligned}$$

978-1-4577-1415-3/12/$26.00 ©2012 IEEE

TABLE I
CORDIC-FUNCTIONS

μ[1]		Rotation Mode(RM) $d_i = sign(z_i)$	Vectoring Mode(VM) $d_i = -sign(x_i y_i)$
1	X_n	$K(x\cos z - y\sin z)$	$K\sqrt{x^2+y^2}$
	Y_n	$K(y\cos z + x\sin z)$	0
	Z_n	0	$z + \tan^{-1}(y/x)$
0	X_n	x	x
	Y_n	$y + xz$	0
	Z_n	z	$z + y/x$
-1	X_n	$K'(x\cosh z - y\sinh z)$	$K'\sqrt{x^2-y^2}$
	Y_n	$K'(y\cosh z + x\sinh z)$	0
	Z_n	0	$z + \tanh^{-1}(y/x)$

[1] $\mu = \begin{cases} 1 & \Theta_i = \tan^{-1} 2^{-i} \quad \text{Circular} \\ 0 & \Theta_i = 2^{-i} \quad \text{Linear} \\ -1 & \Theta_i = \tanh^{-1} 2^{-i} \quad \text{Hyperbolic} \end{cases}$

The accuracy of CORDIC arithmetic contains two parts. Firstly, the theoretical algorithm results are based on infinity iterations and limited recursive steps will result in precision problem. However, we take Circular mode as illustration, when i is large enough, we have $\tan^{-1} 2^{-i} \approx 2^{-i}$. Therefore, for $n > i$, the change in Z will be less than one ulp. So n step iteration will result in n bits accuracy. Linear mode are the same situation. But Hyperbolic mode will give the same accuracy and convergence, only if we repeat step $4, 13 \ldots, n, 3n+1$ once more. Another part of the accuracy part is related with the finite storage bits length. Considering that the truncation of the input arguments after L iterations gives rise to at most a total of $\log_2 L$ (L denotes the bit length of storage for each variable) bits of error. Therefore, we can use $L + \log_2 L$ bits [5] as the length of storage to omitted the regards of harmless.

C. Algorithm for Complex Division & Square-Root

Thus far, very limited research was on the topic of complex divider and square-root. Edman [6] proposed a simple design of standard complex division formula with pipelined real operations. Liu [7] used the DCD algorithm in the complex division without using multiplication and divider. Wang [8] designed a high-radix complex divider with optimized Recurrence method.

Complex division is usually implemented in software-level based on its basic formula,

$$z = \frac{a+ib}{p+iq} = \frac{ap+bq}{p^2+q^2} + i\frac{bp-aq}{p^2+q^2}$$

However, this formula may lead to overflow in computing the intermediate variable $p^2 + q^2$. Smith [9] gave another more robust formula to avoid the overflow issues. Both two parts of the fraction are divided by the larger one between p and q. But this schema would require very costly hardware implementation.

An algorithm for complex division based on CORDIC, suitable for hardware implementation, has been introduced by Sin [10]. The algorithm are simplified and implemented to obtain $(a+ib)/(p+iq)$. The Algorithm1 shown below gives the procedure details.

Algorithm 1: 1. Convert into polar form.

(i) Initialize value as

$$X_0 = p, \quad Y_0 = q, \quad Z_0 = 0$$

(ii) Operate CORDIC of CV(circular vectoring) mode, and we have

$$X = K\sqrt{p^2+q^2}, \quad Y = 0, \quad Z = \tan^{-1}(p/q)$$

(iii) We refer the previous result X and Z by r and θ

(iv) Take the input value as bellow and operate CORDIC of LV(linear vectoring) mode, and we restore the Z output a/r

$$X_0 = r, \quad Y_0 = a, \quad Z_0 = 0$$

(v) Repeat the step(iv) with the follow input and we will get b/r at the output of Z

$$X_0 = r, \quad Y_0 = b, \quad Z_0 = 0$$

2. Rotate back.

(vi) Take another CORDIC of CR(circular rotation) mode with following inputs which we have already get above,

$$X_0 = a/r, \quad Y_0 = b/r, \quad Z = -\theta$$

(vii) Finally, we will get the real and imaginary parts of quotient X and Y respectively.

$$X = (ap+bq)/(p^2+q^2)$$
$$Y = (bp-aq)/(p^2+q^2)$$

We should notice that the modification K has been canceled out. We simplified the procedure and constrain the input domain within a range located in the first quadrant. Thus, all the inputs for this algorithm are greater than zeros. For complex numbers outside the preceding range, we can easily convert and scaling the problem to one that is within the domain we defined.

Similar to complex division, complex number square-root operations is always implemented in software-level, such as Hull [11]. These software-level operations take much longer time than a single arithmetic instruction. Implementation of complex number square-root at the hardware level are quite rare. Ercegovac [12] proposed a recurrence method of complex square root with operand prescaling in hardware-level. Wang [13] present a design for FPGA implementation of a complex square root algorithm for fixed-point operands in radix-4 representation. The conventional implementation of complex square-root is based on the algorithm below,

$$\sqrt{x+iy} = \sqrt{\frac{\sqrt{x^2+y^2}+x}{2}} \pm i\sqrt{\frac{\sqrt{x^2+y^2}-x}{2}}$$

The implementation of this arithmetic requires 2 multiplications (for absolute value computation) and 3 square-roots. This equation above also have the problem of intermediate overflows, and Kahan [14] provided an alternate method to avoid it. Nonetheless, this avoidance will cost more computation including several tests.

Sin [10] proposed an arithmetic for complex number square-root based on CORDIC. As complex number divider, we constrain the input domain within the first quadrant quad and simplified the algorithm for FPGA implementation. Algorithm for $\sqrt{p+qi}$ is shown as below,

***Algorithm** 2:* Step (i) and (ii) are the same as in Algorithm 1.

(iii) We refer the previous result as,

$$r = Z \quad \theta = Z/2$$

(iv) Take the input value as bellow and operate CORDIC of HV mode,

$$X_0 = r + c, \quad Y_0 = r - c, \quad Z_0 = 0$$

where, $c = 0.32649838486$

(v) Shift the previous X result one bit right, and take the result as the X_0 of the next CORDIC, as well as the previous θ as Z_0

$$X_0 = X/2, \quad Y_0 = 0, \quad Z_0 = \theta$$

(vi) After operate CR mode CORDIC with the input given by last step, we will get one of the quare roots,

$$X = \sqrt{(p + \sqrt{p^2+q^2})/2},$$

$$Y = \sqrt{(-p + \sqrt{p^2+q^2})/2},, \quad Z = \theta$$

Another square root will be given by $-X - Y$. Elaborately choosing the constant C can cancel the modulus and enlarge convergence domain. Since the final modulus of the Algorithm 2 comes from $KK'2\sqrt{cKr}$, we can choose $c = 1/(4K'^2K^3)$, which make the modulus equals 1. However, this constant c will result in convergence problem. The convergent condition of HV mode CORDIC is, $|\tan^{-1}\frac{t-c}{t+c}| < 1.118\cdots$.It can be rewrote in the form of $0.1068 < t/c < 9.36$. IF we use the c before, the convergence interval will be $0.0053 < r < 0.4635$, which is too small to include our input domain. So we use four times of constant c, which sure cover our defined interval and also explain the one bit right shift. This complex number square-root requires three CORDIC, while the conventional formula needs two multiplication and three square-roots.

III. DESIGN ARCHITECTURE

A. Implementation of fixed-point CORDIC unit

There are a number of ways to implement CORDIC core [15]. The ideal architecture depends on the tradeoffs between speed versus area. Considering that the most extensive computerized use of complex number division and square root

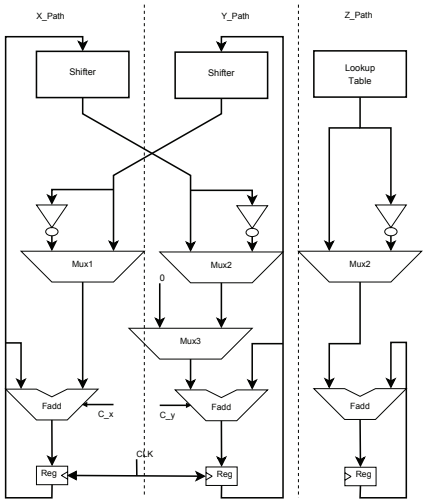

Fig. 1. Diagram of CORDIC

are independent. We want to minimize the area of each unit and duplicated the arithmetic unit to compensate the needs of massive calculation.

Here we use the 16 bits precision CORDIC core, which can be easily extend to 32 bits or more. A block diagram of the single CORDIC core depicted in Figure 1 which consist one lookup table, three of 22-bit register, three 22-bit full adder and two barrel liked shifter.

The CORDIC core unit accepts 18-bit fix-point input, which consist one bit sign, another bit integer and 16 bits binary fraction. The internal bit parallel data path use 4 more bits to render the error bringed by the finite length of storage. The CORDIC unit are iterated 16 times and the accuracy is only limited by the truncation of input arguments. Instead of the subtract operation, an invertor is used and followed with a full adder, where the one more bit to plus is considered as the carry bit. The decision signals for Mux is driven by the sign bits of x, y, z registers and also depend on CORDIC mode. As depicted in Figure 2, Barrel liked shifter are used to meet the needs of 1 to n bits right shifter, and the decision signal for these Mux are exactly each bit of the binary number needs to shift.

In operation, the initial operands of X, Y, Z are loaded into each registers via multiplexes. Then on each of the following 16 clock cycles, the values from the registers pass through the shifters, multiplexers and full adders, then the result are stored back to the registers. The shifter should perform the correct bits right shifts on each iteration. Like wise, the address

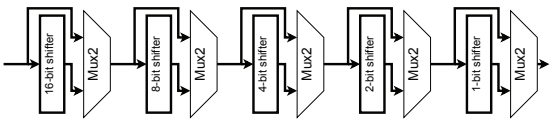

Fig. 2. Scheme of Shifter

TABLE II
PERFORMANCE OF CORDIC UNIT

Combinational ALUTs	270
Regitsters	76
Max frequency	220Mhz
Largest r2r time	4.366ns
Dynamic thermal power	29.65mW
Static thermal power	304.21mW

for lookup table is incremented on each step so that the appropriate elementary angle value is present to the Z adders. On the last iteration, the result are stored into the register. This unit retains the result of each executed mode and function.

Table II provides performance details of the implement for CORDIC core with Altera Stratix II FPGA.

B. Implementation of complex number division

Based on single CORDIC core unit, the Algorithm1 and Algorithm2 in section(ii) can be implemented. The scheme of complex number division contains three main parts: prescaling, recurrence evaluations and postscaling. The input domain of division we defined for both numerator and denominator is $\frac{1}{2} \leq \|x+iy\|_\infty \leq 1$. The complex number, which outside this domain, can be converted into the defined domain by finite binary shifts.

The design diagram in Figure 3 use 18-bit length datapath(one for sign, another for integer). The complex number division contains four CORDIC units. The CV mode unit transfer the denomination into polar coordinate. Two CORDIC unit in LV mode perform as the real division. Finally the CR mode unit convert back to Cartesian coordinate.

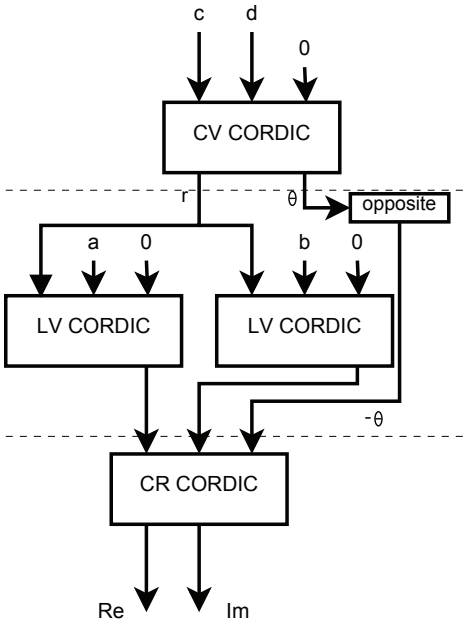

Fig. 3. Scheme of division

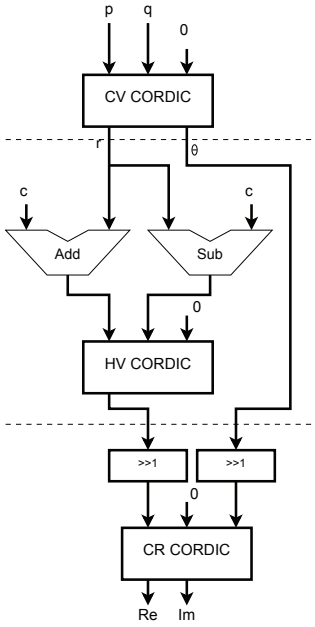

Fig. 4. Scheme of square-root

These two LV mode CORDIC units share the same lookup-table, due to they are working synchronized, as well as two circular mode units. If these two LV mode units are operated parallel, it will cost 48 cycles to gain the division result. And it will cost 16 cycles more, if we use a generally CORDIC unit to replace these four units, for the optimized area consuming. In operation, the real and imaginary part of denominator are loaded into circular vectoring unit. After 16 clock periods, its modulus is passed into the following two linear vectoring unit. And the additive inverse of angle is as the input of the rotation back unit. The four cordic units share the same clock. These three layered architecture is easily pipelined to increase the throughput by inserting registers between each layer interface.

C. Implementation of complex number square-root

As described in Figure 4, the implementation of complex number square-root consist two extra adders, two 1-bit right shifters and three CORDIC units. These three units are in CV, CR and HV mode. The CV and CR mode units can share the same lookup-table, and the HV mode CORDIC does not need the lookup table at all. The reason is that the square-root arithmetic only need the X output of HV unit, and decision circuit under vectoring mode is not related with the angels.

The full scheme of complex number square-root contains three main parts: prescalingrecurrence evaluations and postscaling. The defined input domain of Complex number square-root is $\frac{1}{2} \leq |x+iy| \leq 1$. For complex number outside, they can be converted and scaled into the defined domain.

This architecture accept 18-bit word length input, and the datapath between each CORDIC unit is also 18-bit. The input are loaded into CV unit and converted into polar form. After

TABLE III
PERFORMANCE OF DIV & SQRT

Complex number operation 16-bit	Division	Square-root
Combinational ALUTs	804	700
Regitsters	269	201
Max frequency	180Mhz	188Mhz
Largest r2r time	5.17ns	5.13ns
Dynamic thermal power	55.85mW	54.39mW
Static thermal power	304.15mW	304.08mW

plus and subtract with a constant c, the modulus of $p + iq$ are regarded as the input of HV CORDIC. After 18 clocks iteration, two times fourth root of modulus can be obtained at the X output. Therefore, putting the half of HV result and theta at the input of rotation back unit, we can have the result of step (vi) in Algorithm2.

This Square-root needs 50 cycles(two more cycles for HV mode convergence) to obtain the roots result. The one bit right shift are easily implemented by repeating the sign bit and omit the LSB. The individual subtract unit are same as the adder, since we use both of the constant c and its two's complement.

A unbiased round-to-nearest result is shown in Table III. These designs run at the maximum speed of 180Mhz and 188Mhz. And their resources consuming are ultra-small to implemented with current FPGA. The complex operand have 16 bits of precision for real and imaginary parts respectively. The proposed design is implemented in Verilog HDL, verified by functional simulation, synthesized, and mapped in Altera Stratix-II FPGA.

IV. CONCLUSION

In this paper, we have presented efficient hardware implementations of complex number divider and square-root with good numerical properties. Due to its area and energy efficiency, the scale of implementation could be tens or hundreds to meet the needs of mass complex number computation in engineering projects. This design are well fitted to implemented as arithmetic co-processor and computation units in SoC. These architects are only three-Level pipelining. Therefore, the future work could focus on more level pipelining to increase the throughput.

ACKNOWLEDGMENT

The authors would like to thank the reviewers and the associate editor for the helpful comments that greatly improved this brief.

REFERENCES

[1] K.-I. Ko and F. Yu, "On the complexity of computing the logarithm and square root functions on a complex domain," *Journal of Complexity*, vol. 23, no. 1, pp. 2–24, 2007.

[2] M. Sima, M. Senthilvelan, D. Iancu, J. Glossner, M. Moudgill, and M. Schulte, "Software solutions for converting a mimo-ofdm channel into multiple siso-ofdm channels," in *In Proceedings of the 3rd IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, oct 2007, p. 9.

[3] J. Xiang, L. Guo, Y. Chen, and J. Zhang, "Study of gps adaptive antenna technology based on complex number aaca," in *IEEE International Conference on Wireless Communications, Networking and Mobile Computing,*, oct 2008, pp. 1–4.

[4] J. VOLDER, "Binary computation algorithms for coordinate rotation and function generation," Convair Report IAR-1 148 Aeroelectronics Group, 1956.

[5] J. S. Walther, "A unified algorithm for elementary functions," in *Spring Joint Comp*, ser. Conference 1971, 1971, pp. 379–385.

[6] F. Edman and V. Owall, "Fixed-point implementation of a robust complex valued divider architecture," in *Proceedings of ECCTD05*, Cork, Ireland, Aug 2005.

[7] J. Liu, B. Weaver, and Y. Zakharov, "Fpga implementation of multiplication-free complex division," *Electronics Letters*, vol. 44, no. 2, pp. 95 –96, 17 2008.

[8] D. Wang, M. Ercegovac, and N. Zheng, "A radix-8 complex divider for fpga implementation," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, Aug 2009, pp. 236 –241.

[9] R. Smith, "Algorithm 116: Complex division," Communications of the ACM, 1962, 5(8):435.

[10] S. Hitotumatu, "Complex arithmetic through cordic," *Kodai Math.SEM.REP*, vol. 26, no. 2-3, pp. 176–186, 1975.

[11] T. E. Hull, T. F. Fairgrieve, and P.-T. P. Tang, "Implementing complex elementary functions using exception handling," *ACM Trans. Math. Softw.*, vol. 20, no. 2, pp. 215–244, jun 1994.

[12] M. D. Ercegovac and J.-M. Muller, "Complex square root with operand prescaling," in *Proceedings of the Application-Specific Systems, Architectures and Processors, 15th IEEE International Conference*, ser. ASAP '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 52–62.

[13] D. Wang and M. Ercegovac, "Design of high-throughput fixed-point complex reciprocal/square-root unit," *Circuits and Systems II*, vol. 57(8), pp. 627–631, 2010.

[14] W. Kahan, "Branch cuts for complex elementary funcions, or much ado about nothing's sign bit," in *The State of the Art in Numerical Analysis*. Oxford: Clarendon Press, 1987.

[15] 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*. New York, NY, USA: ACM, 1998, pp. 191–200.