低时延低消耗自由扩展CORDIC算法及结构研究

任小西 刘 明
(湖南大学信息科学与工程学院 长沙 410082)

摘 要 自由扩展CORDIC算法以其计算一些特定函数的能力为我们所熟知。但是有限的适用区间和较慢的速度成为其重要的缺点。虽经大量改进,CORDIC算法在适用区间和执行速度上仍面临较大的挑战。提出的方案通过误差校正的全局自由扩展机制来使收敛域虚拟地达到足够大的区域,且使用降位迭代计算方式并行化加速计算。通过仿真、综合可以看出,与改进的版本相比,得到的新结构改善了有限的收敛域,降低了一个时钟的时延,减少了25.5%的硬件消耗以及35%的功耗。

关键词 CORDIC算法,低时延,低消耗,自由扩展
中图法分类号 TP302.2 **文献标识码** A

Research on Low-latency and Low-consumption Scaling-free Algorithm and Architecture

REN Xiao-xi LIU Ming
(Department of Imformation Science and Engineering, Hunan University, Changsha 410082, China)

Abstract Scaling-free CORDIC algorithm is well known by its ability of calculating special function. Its limited range of convergence and lower speed are an important drawback. There is still some progress to make, although some new version has been proposed. Comparing with modified original scaling-free CORDIC algorithm, our proposed algorithm can virtually converge to a bigger range, because of adding a correction iteration, what is more, accelerate computation though parallelization. After simulation and synthesizing, we can obtain some new architectures which are able to reach one clock lower latency and a 25.5% reduction in area and 35% reduction in power consumption compared to the modified original scaling-free architecture.

Keywords CORDIC algorithm, Low-latency, Low-consumption, Scaling-free

1 引言

CORDIC(坐标旋转数字计算机)算法[1,2]是一个循环迭代算法,该算法的基本思想是通过一系列固定的、与运算基数有关的角度不断偏摆迭代从而逼近所需要的旋转角度,只需使用移位加操作。该算法能在线性、圆周和双曲坐标中对二维矢量进行旋转,本文在圆周坐标中进行。CORDIC算法应用范围很广,如数字信号处理领域和矩阵运算[3,4]、频率合成[5,6]、三角函数运算[7,8]、矩阵分解[9,10]、三维旋转[11]、无线通信[12]等等。由于一些缺点的存在,持续有研究对CORDIC算法进行改进。如针对迭代导致的速度较慢问题,文献[14]在规模因素方面提出了修正方案,文献[15]提出了有效的CORDIC算法和结构以实现低面积和高吞吐量。针对扩展因子的补偿,也有多种改进方法及如文献[16]中描述的并行方法、文献[17]中展示的快速整合,此方法需要一个额外的时钟周期,改进后最终获得了无扩展因子向量。在文献[18]中,作者提出了一个为无线通信设计的自由扩展因子CORDIC算法,其已成功应用在基于OFDM的无线局域网系统的内部接收机中。此CORDIC算法能够避免产生变化的扩展因子,只要计算必要的微旋转,与传统CORDIC算法相比,减少了旋转次数。目前已经改进的自由扩展因子CORDIC算法仍然存在一些缺点,如采用的是全部流水结构的设计方式,或者为了精度又增加了迭代的次数,另外一个缺点是自由扩展因子的适用范围比较小。

本文提出了改进的自由扩展因子CORDIC算法,其在算法级和结构级进行了一些改进,将其限定在$\theta<\pi/4$范围内,可以减小误差,而其他范围内的角度也可以很容易转换到这个范围内。带有校正迭代的全局Scaling-free,通过部分并行计算、降低运算位数等等,减少了迭代次数、运算的功耗,同时也提高了运算的速度。

2 传统CORDIC算法与自由扩展因子CORDIC算法

2.1 传统CORDIC算法

CORDIC算法非常适合在FPGA上实现,因为CORDIC算法的主要思想是通过迭代实现向量的旋转,其基本原理如图1所示。

从式(1)可以看到,如果向量(X_0,Y_0)转移角度θ,就可以得到一个新的向量(X_1,Y_1)。

到稿日期:2012-03-01 返修日期:2012-07-29 本文受国家自然科学基金(61173037),湖南大学"青年教师成长计划"项目资助。
任小西 女,博士生,副教授,主要研究方向为可重构计算系统与多核计算系统,E-mail:happyrxx@163.com;刘 明(1986-),男,硕士生,主要研究方向为可重构计算系统以及多核计算系统。

$$\begin{cases} x_1 = x_0 \cos\theta - y_0 \sin\theta \\ y_1 = y_0 \cos\theta + x_0 \sin\theta \end{cases} \tag{1}$$

重新整理后得式(2)：

$$\begin{cases} x_1 = \cos\theta[x_0 - y_0 \tan\theta] \\ y_1 = \cos\theta[y_0 + x_0 \tan\theta] \end{cases} \tag{2}$$

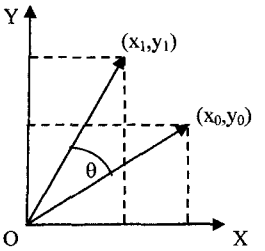

图1 传统CORDIC算法原理图

为了方便硬件执行，设定了 N 次旋转，并且每一次旋转角度 θ_i，而 θ_i 对应 $\tan\theta_i = 2^{-i}$，所以 $\cos\theta_i = \sqrt{\frac{1}{1+2^{-2i}}}$，第 i 次的旋转如式(3)所示。

$$\begin{cases} x_{i+1} = (x_i - \delta_i y_i 2^{-i})\sqrt{\frac{1}{1+2^{-2i}}} \\ y_{i+1} = (y_i + \delta_i x_i 2^{-i})\sqrt{\frac{1}{1+2^{-2i}}} \\ z^{i+1} = z^i - \delta_i \tan^{-1} 2^{-i} \end{cases} \tag{3}$$

在第 i 次旋转后，角度变换是 Z_i，每一次旋转的方向是 δ_i，它等于 Z_i 的符号，也就是 $\delta_i = \mathrm{sign}(Z_i)$。当 $\delta_i = +1$ 时，朝逆时针旋转，当 $\delta_i = -1$ 时，顺时针旋转。$\sqrt{\frac{1}{1+2^{-2i}}}$ 是每一级的校正因子，对于定长的处理单元，整个 $\sqrt{\frac{1}{1+2^{-2i}}}$ 的校正因子是确定的，如果全部的旋转序列是 N，最后的校正因子用 K 来表示，如式(4)所示：

$$K = \prod_{i=0}^{N-1} \sqrt{\frac{1}{1+2^{-2i}}} \tag{4}$$

以16位为例，$K = 0.607252935$。

可以在第一次旋转操作时就校正数据，所以每一级操作可以简化成式(5)。

$$\begin{cases} x_{i+1} = (x_i - \delta_i y_i 2^{-i}) \\ y_{i+1} = (y_i - \delta_i x_i 2^{-i}) \\ z_{i+1} = z_i - \delta_i \tan^{-1} 2^{-i} \end{cases} \tag{5}$$

2.2 自由扩展CORDIC算法

与传统CORDIC算法不同，在自由扩展CORDIC算法中，目标角度也是通过旋转原来的向量达到的，但是只朝一个方向旋转，从而避免了一些不必要的旋转。这就意味着目标角度约为许多单位角度的纯加运算。这个新的算法 $\sin\theta_i$ 和 $\cos\theta_i$ 近似如式(6)所示的形式。

$$\begin{aligned} &\sin\theta_j \cong 2^{-j} \\ &\cos\theta_j \cong 1 - 2^{-(2j+1)} \end{aligned} \tag{6}$$

但是，这个公式只能适用的范围如式(7)所示：

$$\left[\frac{n-2.585}{3}\right] \leqslant j \leqslant n-1 \tag{7}$$

式中，n 代表字长。

自由扩展因子CORDIC算法同样也解决了角度旋转通路 Z 的问题，每一次迭代只需要输入角度的对应位来决定单次微旋转是执行还是跳过。因此，自由扩展因子CORDIC算法迭代公式在顺时针情况下可以如式(8)所示：

$$\begin{aligned} x_{j+1} &= (1 - 2^{-(2j+1)})x_j - 2^{-j} y_j \\ y_{j+1} &= (1 - 2^{-(2j+1)})y_j + 2^{-j} x_j \end{aligned} \tag{8}$$

结果，自由扩展因子CORDIC迭代也可以通过移位加操作来完成。另外，当 $j \geqslant n/2$ 时，输入角度向右移动 $2j+1$ 位相当于等于0，公式中负责计算 $1 - 2^{-(2j+1)}$ 的移位和减法器不再需要，因此迭代的硬件块就小一些。这块较简单硬件与传统CORDIC迭代的 X 和 Y 通路差不多。

根据公式，输入的向量长度越大，得到的收敛区间就越小。为了解决这个问题，文献[18]的作者设计了两种不同的技术来保证收敛区间扩展到整个坐标空间，这两种方法是：

1)自由扩展因子CORDIC算法第一次单位微旋转重复多次，直到 θ_{max} 到达 $\pi/8$ 范围内。

2)域重载技术，把 $[0,\pi/8]$ 区间扩展到 $[0,\pi/2]$。

然而，为了让收敛域达到 $\pi/8$ 而进行第一次单位微旋转的重复有一个重要的缺点。例如，随角度或者是字长的提升，重复迭代的数量就会增加，导致硬件消耗大于传统CORDIC算法一次完成的。

域重叠技术通过在自由扩展因子核心之前的前处理模块实现，把最初的角度 θ 属于 $[0,\pi/2]$ 降低到较小的角度 ϕ 属于 $[0,\pi/8]$，这样就便于自由扩展因子核心来实现。有时，域重载技术可能会引入一个 $1/\sqrt{2}$ 的扩展因子，这取决于输入角度 θ 所在的域。然而，在必要的时候，它可以由后处理模块进行校正，故扩展因子对于自由扩展因子核心来说是简易的。

式(7)表示了自由扩展因子CORDIC算法的一个真实的限制，就硬件需求方面来讲，当字长达到20位的时候，还不如使用传统CORDIC算法来实现。然而，在相当数量的16位字长应用方面，自由扩展因子CORDIC算法效果仍然非常明显。在这个情况下，自由扩展因子CORDIC旋转器与传统CORDIC算法相比，在面积(降低了19%的加法器和53%的寄存器)和速度方面有明显的性能提升。

另外，在文献[18]中，作者实现了一个16位的自由扩展因子CORDIC算法，14位最低有效位作为小数部分，在经过域重载技术后，$\phi \leqslant \pi/8$，ϕ 只需要13位来表示。另外，θ 需要18位来表示。

整个自由扩展因子CORDIC的结构是：一个前置处理模块用来处理象限和域探索以及进行域重载技术；自由扩展因子核心计算式(8)；后置处理模块通过输入角度的象限和域来保证正确的输出值并且在必要的时候补偿 $1/\sqrt{2}$。

3 改进的自由扩展CORDIC算法

自由扩展因子CORDIC算法收敛区间较小，虽然可通过重复第一次微旋转迭代来达到效果，但是当输入角度和字长增加时，重复的次数就成倍增长了，这就增加了算法的迭代次数，降低了运算的速度。另外，后半部分算法虽然简化了，但是速度并没有因此得到大的提升。针对这些不足，本文提出了如下两点改进。

3.1 全局自由扩展机制

因为旋转角度 θ_i 与对应该位置二进制权值 2^{-i} 近似相等，所以可以根据输入角度 θ 的二进制表示直接推导出每次基本迭代的旋转方向。但是出于误差的控制，这只能在一定范围内适用，也就是在收敛区间内有效，在非收敛区间内就只能用传统CORDIC算法来计算或者是借助收敛区间的第一

个基本迭代的重复执行来近似完成。这样不是加大了运算的复杂度，就是增加了迭代的次数，都不可取。

鉴于如上缺点，将式(8)的适应范围扩大至较大的角度。出于误差的控制只适用于迭代索引 $i>[(N-2.585)/3]$，为了让迭代索引在 $i<[(N-2.585)/3]$ 区间，在迭代序列中加入校正迭代，以保证最终旋转角度的正确性。下面来推导一下应该在什么位置加入校正迭代。因为输入角度不确定，所以哪些基本旋转执行也不确定，我们先根据所有旋转都要执行来推导出哪些位置需要进行校正，然后根据实际情况设定一个调整机制来实时满足各种不同角度的校正要求。

全部旋转的情况下，因为 $\theta=b_0,b_1\cdots\cdots b_{j-1},b_j\cdots b_k$，这样每一次迭代产生的角度误差就为 $2^{-i}-\theta_i$，规定 $k-j+1$ 次迭代产生的总误差 e 必须小于 2^{-i-1}，即如式(9)所示：

$$\sum_{i=j}^{k}|2^{-i}-\alpha_i|\leqslant 2^{-k} \tag{9}$$

公式给出了满足迭代收敛条件的 j 和 k 的关系，如表 1 所列。

表 1　j 和 k 之间的关系表

j	0	1	2	3	4	5	6	7	8	9	10
k	1	5	8	11	14	17	20	23	26	29	32
j	11	12	13	14	15	16	17	18	19	20	21
k	35	38	41	44	47	50	53	56	59	62	65

在全局旋转的时候可以根据表 1 确定在什么位置加入校正迭代，使得增加的迭代次数最少，而且所用硬件资源最少。例如，对于 32 位来说，增加校正迭代 1、5、10 就可完成补偿。但是本文中的自由扩展因子 CORDIC 算法的旋转位置不确定，因此必须根据输入角度来确定在哪些位置进行校正，然而不能完全根据式(9)计算出需要校正的位置，故对此进行了一些改进。可以制定一个自动调节的机制，这个机制的制定需要通过全部旋转计算得出对应的收敛区间。

首先，根据输入角度的第一个旋转的位置，确定后续需要旋转的位置，而不需要考虑在后续需要旋转的位置上是否需要旋转，原因是产生误差的主要是第一个旋转，直到最后一个校正位置时，只需要校正这个位置而不需要再进行校正迭代。例如，对于 32 位来说，如果第一次迭代的位置是 1，那么可以增加校正迭代 5、10，校正迭代 10 就可以确保误差控制到最后一位，所以对于确定字长的算法中，最后一个校正的位置可以固定。这样就不用再进行动态判定。考虑到每一次实际旋转迭代的角度要大于理论上需要旋转的角度，所以加入的校正迭代与总体的旋转方向相反。

3.2　降位迭代计算

全局自由扩展计算完成了自由扩展因子 CORDIC 算法的前半段数据的处理。后半段数据的计算公式简化了，因为后半段数据都要移位 $N/2$，所以式(8)中的 $2^{-(2j+1)}$ 基本可以省略，因为 N 位的数移位 N 位后就为 0 了，这样，后半段的计算公式可以化简为如下形式：

$$\begin{aligned} x_{j+1}&=x_j-2^{-j}y_i \\ y_{j+1}&=y_i+2^{-j}x_i \end{aligned} \tag{10}$$

后半段仍然采用文献[18]中的两两联合的方式，以减少迭代级数，针对后半段的资源消耗，在后半段做了一些改进。后半部分的运算都要移位 $N/2$ 以上，且实际参与移位累加运算的只有每个数据的 $N/2$ 的高位数字，在此处对数据进行处理，将数据都减少 $N/2$ 位进行计算，在输出时再将数据进行处理，补全数据。例如，对于 32 位字长的数据，将其分为高 16 位和低 16 位，高 16 位一方面作为最后补全 32 位数据的高 16 位，另一方面，作为移位累加运算的基数，如原来需要移位 20 位，现在只需要移位 4 位，全部移位后的数据进行累加之后再加上原来数据的低 16 位，这里需要增加 1 位，使用 17 位运算，以防止数据溢出，然后将移位累加并加上低 16 位的结果作为一个 32 位数据的低 17 位，再加上以高 16 位作为一个 32 位数据的高 16 位的数据，得到最终结果，这样就降低了硬件资源的消耗。公式如下：

$$\begin{aligned} x_{NL}&=x_{iL}-y_{iH}2^{-j} \\ y_{NL}&=y_{iL}+x_{iH}2^{-j} \\ x_{NH}&=x_{iH} \\ y_{NH}&=y_{iH} \end{aligned} \tag{11}$$

4　实现及数据比较分析

所有的结构用 Verilog HDL 来实现，使用 Mentor Graphics Modelsim SE 来仿真，用 ISE 10.1 进行综合。

为了更好地评估改进的 CORDIC 旋转算法和结构，主要分析 3 个方面，即 X 和 Y 的计算误差、完全 CORDIC 算法所需的面积以及运算的时延。主要考虑的方面是提出的算法所需要的硬件数量以及整个运算的速度。除此之外，也将探索在功耗方面的优点。

误差分析：用数字电路执行传统自由扩展因子算法时，存在两个主要的问题：运算时要对旋转角度进行量化处理，然后将角度近似为运算的操作数，这样就比真实的数据要少，另外在实际运行时，由于运算的精度有限造成累积的舍入误差。后者取决于相关旋转值的旋转操作次数。角度近似替换让操作角度小于实际单元角度，全局自由扩展机制让这个误差加大，但是在加入校正迭代后又让这个误差缩小。文献[18]中的算法采用 16 位向量计算，在自由扩展算法运算后，得到上限为 12 位的 X 和 Y 通路的十进制有效位。

本文提出的算法的结构是原始自由扩展因子 CORDIC 算法的改进版本。其主要目的是提高运算的速度以及降低资源消耗。原始自由扩展结构的精度误差如图 2 所示；如图 3 所示，误差基本控制在十进制数的 12 位的周围。

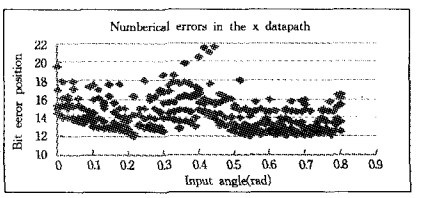

图 2　原始自由扩展结构的精度误差

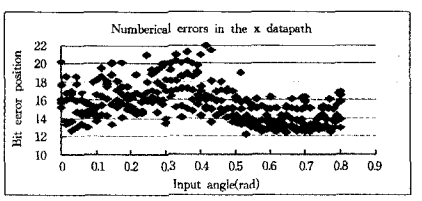

图 3　改进的自由扩展结构精度误差

面积需求(见表 2)：改进的 CORDIC 旋转器结构中，同样包括前端的符号与象限探索单元和后置纠正输出单元，以及最基本的 CORDIC 流水结构。前端符号与象限探测单元需要两个 16 位加法器和两个比较器，每个这种比较器相当于一个 4 位的加法器。因为改进的方案是将输入角度范围控制在

[0,π/4]范围内，输出纠正单元就只需要实现数的交换和取相反数，没有了加法器和扩展因子单元。另外，基本 CORDIC 流水结构有 11 节长，前半部分的每一节需要 4 个 16 位加法器，有 6 节常规迭代。后半部分因为化简了降位运算，延用了两两结合的方式进行运算，有 4 节化简后的迭代运算，这个部分每一节需要 4 个 9 位加法器，另外还要加上 1 个校正迭代的一节，这个校正迭代位置不定，可能在前半部分，也可能在后半部分，故可以用一节有 4 个 16 位加法器的迭代进行运算。整个结构大概需要 664 个全加器。

表 2　面积需求

算法结构	等额全加器数目	寄存器总数
传统算法结构	1232	1253
原始自由扩展	1000	723
改进后的结构	664	619

传统的自由扩展因子 CORDIC 算法的实现中，基本的 CORDIC 流水结构是 12 级长，每一级需要 4 个 16 位加法器。符号与象限探索单元需要两个 16 位加法器和两个比较器。每个比较器相当于一个 4 位加法器。输出单元由 2 个 16 位的加法器和两个扩展单元，因为传统的自由扩展因子 CORDIC 算法将输入角度范围限定在[0,π/8]范围内，所以在纠正单元需要一个扩展因子 $1/\sqrt{2}$。这个扩展单元是由 5 个 16 位加法器组成的。传统的算法执行时需要的最高面积是 62 个 16 位加法器和两个 4 位加法器，大概等于 1000 个全加器。

时延：本文所提出的方案中，基本的 CORDIC 旋转流水线是 11 级，再加上符号与象限探索以及输出纠正单元，整个 CORDIC 流水级长度是 13 级长。处理器的时延是 13 个时钟周期。本文采用的方案不用多次旋转第一个收敛区域的迭代来达到较大角度的旋转，而是直接对较大角度的旋转与收敛区间的旋转角度采用同样的处理，虽然加入了校正迭代，但最终的结果减少了迭代的次数，同时减少了迭代级数。从而整体提高了 CORDIC 处理器的运算速度。

功耗：使用 ISE 自带的 Xpower 工具来进行功耗分析。对于各种结构的结果，从表 3 中可以得到清晰的比较。

表 3　功耗

算法结构	100MHz	700MHz
传统算法结构	1.710	12.513
原始自由扩展	1.426	10.699
改进后的结构	0.962	7.053

可以看到，随着迭代次数的减少以及流水级数的减少，功耗降低了。本文所采用的方案，并不是每一级的迭代都会进行，跳过了不必要的角度，在功耗方面，各个不同的运算频率上平均降低了 35%左右。另一方面，本方案极大地减少了加法器和寄存器的数量，进一步降低了功耗。

结束语　最初的自由扩展因子 CORDIC 算法将收敛区间限定在 π/8 以内，而本文为了减少前置和后置处理的负担，将收敛区间扩大到 π/4 以内，由此带来的误差通过后续处理中在合适的位置加入校正迭代来进行纠正。另外，针对后半部分运算速度较慢且消耗的硬件资源较多的情况，本文通过化简后半部分，使之可以进行并行计算，减少了流水级数，同时也加快了运算的速度，且通过降位运算，减少了约为一半的硬件消耗。本文所提出的方案相较于最初的自由扩展因子 CORDIC 算法，在面积、时延以及功耗方面都有所改进。在第 4 节中可以看到，新的结构经过仿真、综合相较于最初的结构少了一个时钟的时延，功耗也降低了约 35%，硬件消耗降低了 25.5%。最终得到的结果也并没因此出现较大的误差，与最初的自由扩展因子 CORDIC 结构处理器处于相同的数量级。

参 考 文 献

[1] Volder J E. The CORDIC trigonometric computing technique [J]. IRE Trans. Electron. Comput., 1959, 8(3): 330-334

[2] Hu Y H. CORDIC-based VLSI architectures for digital signal processing[J]. IEEE Signal Process, 1992, 9(3): 13-35

[3] 孙震，陆斌斌. 基于混合 CORDIC 算法的 DDFS 的研究[J]. 电子测量技术，2009(7): 58-61

[4] Zapata E L, Arguello F. A VLSI constant geometry architecture for the Hartley and Fourier transform[J]. IEEE Trans. Parallel Distrib. Syst., 1992, 3(1): 58-70

[5] 张海涛，苗圃，庞永星，等. 改进型 Cordic 高精度 DDS 硬件实现[J]. 计算机工程，2011, 37(12): 1000-3428

[6] 麻志鹏，沈小林. 基于 CORDIC 算法的正余弦运算的 FPGA 实现 [J]. 电子测量，2011(3)

[7] Hormigo J, Villalba J, Zapata E L. Arithmetic unit for the computation of interval elementary functions[C]// EUROMICRO Conference, 1999. Sep. 1999, 1: 63-66

[8] Zabit U, Bony F, Bosch T. Implementation of optimized trigonometric functions for a self-mixing laser diode displacement sensor under moderate feedback[C]// Sensors, 2008 IEEE. Oct. 2008, 5: 988-991

[9] Ercegovac M D, Lang T. Redundant and on-line CORDIC: Application to matrix traiangularization and SVD[J]. IEEE Trans. Comput., 1990, 6: 725-740

[10] Hu X, Bass S C, Harber R G. An efficient implementation of singular value decomposition rotation transformations with CORDIC processor[J]. J. Parallel Distrib. Comput., 1993, 17: 360-362

[11] Lang T, Antelo E. High-throughput 3D rotations and normalizations[C]// Proc. 35th Asilomar Conf. Signals, Systems Computers. 2001, 1: 846-851

[12] Zhang Guo-ping, Chin F. Multi-antenna WCDMA receiver design with CORDIC, Personal, Indoor and Mobile Radio Communications, 2004 [C]// PIMRC 2004. 15th IEEE International Symposium on. June 2004, 4: 2811-2814

[13] 张建斌，梁芳，刘乃安. 一种改进型 CORDIC 算法的 FPGA 实现[J]. 微电子学与计算机，2010, 27(11)

[14] Sumanasena M G B. A scale factor correction scheme for the CORDIC algorithm[J]. IEEE Trans. Comput., 2008, 57(8): 1148-1152

[15] Vachhani L, Sridharan K, Meher P K. Efficient CORDIC algorithms and architectures for low area and high throughput implementation[J]. IEEE Trans. Circuits Syst. II, Expr. Briefs, 2009, 56(1): 61-65

[16] Villalba J, Lang T, Zapata E L. Parallel compensation of scale factor for the CORDIC algorithm[J]. J. VLSI Signal Process. Syst., 1998, 19(3): 227-241

[17] Sumanasena M G B. A scale factor correction scheme for the CORDIC algorithm[J]. IEEE Trans. Comput., 2008, 57(8): 1148-1152

[18] Maharatna K, Banerjee S, Grass E, et al. Modified virtually scaling-free adaptive CORDIC rotator algorithm and architecture [J]. IEEE Trans. Circuits Syst. Video Technol., 2005, 15(11): 1463-1474