CORDIC算法的优化及实现

刘小宁， 谢宜壮， 陈禾， 闫雯， 陈冬
（北京理工大学 信息与电子学院，北京 100081）

摘 要：为提高坐标旋转数字计算(CORDIC)算法的精度并降低硬件资源消耗，对CORDIC算法收敛性以及旋转序列的选取进行了研究. 针对圆周系统下CORDIC算法的角度覆盖范围、硬件资源和运算精度等问题提出了进一步的优化措施. 利用经过优化后的CORDIC算法，在FPGA中实现了流水线结构的正余弦函数和反正切函数，并把运算精度与硬件资源消耗与Xilinx IP核进行了比较. 比较结果表明该优化算法在提高运算精度的同时能够有效降低硬件资源消耗.

关键词：CORDIC算法；三角函数；精度；硬件资源；FPGA
中图分类号：TN 47 **文献标志码**：A **文章编号**：1001-0645(2015)11-1164-07
DOI：10.15918/j.tbit1001-0645.2015.11.012

Optimization and Implementation of CORDIC Algorithm

LIU Xiao-ning， XIE Yi-zhuang， CHEN He， YAN Wen， CHEN Dong
(School of Information and Electronics, Beijing Institute of Technology, Beijing 100081, China)

Abstract: In order to improve the precision of coordinate rotation digital computer (CORDIC) algorithm and reduce the hardware resource consumption, research on the convergence and the selection of rotation sequence was conducted. Optimization measures were proposed according to angular coverage, hardware resources and computing precision of CORDIC algorithm for circular system. Sine and cosine functions and reverse tangent functions based on pipeline structure were implemented on FPGA, and computing precision and hardware resource consumption were compared with Xilinx IP core. Comparison shows that the optimized algorithm which improves the computing precision can also reduce the hardware resource consumption effectively.

Key words: CORDIC algorithm; trigonometric function; precision; hardware resources; FPGA

CORDIC (coordinate rotation digital computer)算法是J. Voider等[1]于1959年在美国航空控制系统的设计中提出来的. 其基本思想是用一系列只与运算基数相关的角度不断偏摆从而逼近所需旋转的角度，只需要进行移位和加法运算就能实现包括三角函数在内的一些复杂的超越函数. 由于该算法是一种规则化的算法，它满足了硬件对算法模块化、规则化的要求，从而实现硬件与算法相结合的一种优化方案[2].

传统的CORDIC算法在圆周系统下的覆盖范围只有[−99.82°, 99.82°]，并没有覆盖整个圆周. 同时，CORDIC算法具有迭代性，为了提高算法精度，需要反复进行迭代计算或者增加流水线长度. 反复迭代计算节省硬件资源但是运算速度慢，增加流水线长度会造成运算资源的增加. 针对上述问题，各种优化改进方法不断被提出. 增加多次第一级迭代[2]，将算法的覆盖范围扩展到完整的圆周[−π, π]，但是这种方法中校正因子为变量，需要增加额外的资源去调整. 对旋转角度值进行编码以减少旋转次数[3]，这种方法能有效减少迭代次数，但是

收稿日期：2013-08-13
基金项目：国家自然科学基金资助项目(61171194)
作者简介：刘小宁(1987—)，男，博士生，E-mail：liuxiaoning@bit.edu.cn.
通信作者：谢宜壮(1980—)，男，博士，讲师，E-mail：xyz551_bit@bit.edu.cn.

同样会改变校正因子.

本文主要针对圆周系统下 CORDIC 算法的角度覆盖范围、硬件资源和运算精度提出了进一步的优化措施. 利用三角函数之间的相互转化关系和三角函数的周期性,将 CORDIC 算法的计算范围扩展到任意角度;利用定点 CORDIC 算法运算过程中的数据位宽变化趋势减少硬件资源;通过数据格式化间接扩大运算的数据位宽,提高运算精度.

1　CORDIC 算法基本原理

1.1　圆周系统的 CORDIC 算法

CORDIC 算法是一种逼近算法,由一个角度旋转到另一个角度来逐渐逼近目标角. 圆周系统下的 CORDIC 算法的基本迭代关系[4-5]如下

$$\begin{cases} x'_{i+1} = x'_i - \delta_i y'_i 2^{-i}, \\ y'_{i+1} = y'_i + \delta_i x'_i 2^{-i}, \\ z_{i+1} = z_i - \delta_i \theta_i. \end{cases} \tag{1}$$

根据 δ_i 的取值方式不同,CORDIC 算法又分为两种工作模式:旋转模式和向量模式[1].

1.1.1　旋转模式

旋转模式是通过一系列的角度的旋转逼近目标角的工作方式,用于正弦和余弦的运算. 旋转模式的完整方程如下

$$\begin{cases} x'_{i+1} = x'_i - \delta_i y'_i 2^{-i}, \\ y'_{i+1} = y'_i + \delta_i x'_i 2^{-i}, \\ z_{i+1} = z_i - \delta_i \theta_i, \\ \delta_i = \operatorname{sgn} z_i. \end{cases} \tag{2}$$

假设目标角为 θ,经过 N 次旋转迭代之后,$z_N \to 0$,同时

$$\begin{cases} x_N = \left(\prod_{i=0}^{N-1} \cos^{-1}\theta_i\right)(x_0 \cos\theta - y_0 \sin\theta) = \\ (1/k)(x_0 \cos\theta - y_0 \sin\theta), \\ y_N = \left(\prod_{i=0}^{N-1} \cos^{-1}\theta_i\right)(x_0 \sin\theta + y_0 \cos\theta) = \\ (1/k)(x_0 \sin\theta + y_0 \cos\theta). \end{cases} \tag{3}$$

如果取 $x_0 = k$,$y_0 = 0$,则有

$$\begin{cases} x_N = \cos\theta, \\ y_N = \sin\theta. \end{cases} \tag{4}$$

式(4)表明,圆周系统的旋转模式能用于计算正弦和余弦函数.

1.1.2　向量模式

与旋转模式不同,向量模式是通过一系列角度旋转逼近目标向量的工作方式,可用于反正切运算. 旋转模式的完整方程如下

$$\begin{cases} x'_{i+1} = x'_i - \delta_i y'_i 2^{-i}, \\ y'_{i+1} = y'_i + \delta_i x'_i 2^{-i}, \\ z_{i+1} = z_i - \delta_i \theta_i, \\ \delta_i = -\sin y_i. \end{cases} \tag{5}$$

经过 N 次迭代后 $y_N \to 0$,得到

$$\begin{cases} x_N = \left(\prod_{i=0}^{N-1} \cos^{-1}\theta\right)\sqrt{x_0^2 + y_0^2}, \\ z_N = z_0 + \tan^{-1}(y_0/x_0). \end{cases} \tag{6}$$

如果取 $z_0 = 0$,则有

$$z_N = \tan^{-1}(y_0/x_0). \tag{7}$$

式(7)表明,圆周系统下的向量模式可以用于计算反正切函数.

1.2　算法实现结构

由式(2)和式(5)可以看出,CORDIC 算法具有迭代性,每级迭代运算的结构相对一致,可以使用统一的结构来实现. 因此算法可以用反馈结构和流水线结构分别来实现[2-8]. 反馈结构只需要一级 CORDIC 迭代运算单元,不断将输出数据反馈到输入数据,占用资源少,但数据吞吐量很小. 流水线结构每一级 CORDIC 迭代运算都使用独立的运算单元,占用资源较多,但是数据吞吐量大,运算速度快,适合高速数字信号处理.

2　CORDIC 算法在圆周系统下的优化

2.1　角度覆盖范围的扩展

圆周系统下的 CORDIC 算法是由一系列满足 $\theta_i = \arctan(2^{-i})$ 的特殊角组成,表 1 所示为用于 CORDIC 旋转迭代的部分角度.

表 1　迭代的部分特殊角度

Tab. 1　Some special angles of iteration

i	θ_i	i	θ_i
0	45.000	5	1.789
1	26.565	6	0.895
2	14.036	7	0.447
3	7.125	8	0.223
4	3.576	9	0.111

CORDIC 算法收敛的最大角度由数值计算得到

$$\sum_{i=0}^{\infty} \theta_i \approx 99.827°. \tag{8}$$

即 CORDIC 算法的收敛域为[-99.827°, 99.827°],不能覆盖整个圆周范围,这是传统

CORDIC算法的一个缺点. 在CORDIC算法的两种实现结构中,反馈结构可采用重复迭代法[2]解决上述问题,对于流水结构本文采用数学变换法解决这一问题.

圆周系统下的数学变换法将整个圆周划分为4个象限,通过三角函数关系式将整个圆周范围内的输入转换到在角度覆盖范围内的象限内,也就是0象限. 表2所示为4个象限的划分,与传统的象限划分不同.

表2 象限划分

Tab.2 Quadrant division

象限	范围
0	$(-\pi/4,\pi/4]$
1	$(\pi/4,3\pi/4]$
2	$(-3\pi/4,-\pi/4]$
3	$[-\pi,-3\pi/4]\cup[3\pi/4,\pi]$

利用下面的三角函数变换关系式,可以通过简单的加减或者符号变化,将1、2、3象限的数据都转换到0象限.

$$\begin{cases}\sin(\theta' \pm \pi/2) = \pm\cos\theta,\\ \cos(\theta' \pm \pi/2) = \mp\sin\theta,\\ \sin(\theta' \pm \pi) = -\sin\theta,\\ \cos(\theta' \pm \pi) = -\cos\theta.\end{cases} \tag{9}$$

旋转模式下是将输入的角度转换到0象限;向量模式下是将输入的向量(x,y)转换到0象限范围内. 如果去掉第一级迭代,也就是45°旋转,CORDIC算法的收敛域将变成[-54.827°,54.827°],同样包括0象限. 即通过数学转换后可以舍弃第一级迭代,将迭代序列变为$i=1,2,3,\cdots,N-1$.

由于输入数据经过变换,输出数据需要根据对应的象限信息进行恢复,恢复到对应的象限. 数据的恢复同样采用数学变换的方式,是输入变换的一个逆向过程. 数学变换法会带来额外的资源消耗,因此不适用于反馈结构. 但是由于数学变换法相比原始的迭代方法减少了1级迭代,并且相对于重复迭代法来说整体减少了3级迭代,因此数学变换法更加适用于流水线结构.

2.2 硬件资源的缩减

对于同样的运算,采用定点运算方式相比采用浮点运算方式占用资源会少很多[6],因此本文中采用定点运算的方式对CORDIC算法进行实现. 硬件资源的缩减主要是利用CORDIC算法运算过程中的数据变化趋势以及定点运算的特点,对运算过程中数据位宽进行缩减的方式减少资源. 由于旋转模式和向量模式的逼近方式不同,数据变化趋势也不一样,因此资源缩减的方式也有所不同.

2.2.1 旋转模式

旋转模式是通过一系列角度旋转来逼近目标角的工作方式,角度累加器z_i逐渐趋近于零旋转模式下的硬件资源缩减就是利用角度累加器的变化趋势实现的. 将$\theta_i=\arctan(2^{-i})$展开成泰勒级数后推导得出

$$\theta_i=\arctan(2^{-i})<2^{-i}. \tag{10}$$

$$\theta_i<2\theta_{i+1}. \tag{11}$$

由第i级的迭代公式(2)可以得出

$$|z_{i+1}|=||z_i|-\theta_i|. \tag{12}$$

经过数学变换后输入角度范围变为[-45°,45°],即

$$|z_1|\leqslant 45°<2\theta_1=53.12°. \tag{13}$$

代入式(12),得到

$$|z_2|=||z_1|-\theta_1|<|2\theta_1-\theta_1|=\theta_1. \tag{14}$$

结合式(11)可以推导出

$$\begin{aligned}&|z_3|=||z_2|-\theta_2|<|2\theta_2-\theta_2|=\theta_2,\\ &|z_4|=||z_3|-\theta_3|<|2\theta_3-\theta_3|=\theta_3,\\ &\vdots\\ &|z_{i+1}|=||z_i|-\theta_i|<|2\theta_i-\theta_i|=\theta_i.\end{aligned} \tag{15}$$

结合式(10),可以得出

$$|z_{i+1}|<\theta_i<2^{-i}. \tag{16}$$

也就是说定点化后的θ_i和z_i的有效位都是在不断减少. 那么对于旋转角z_i的运算,运算位宽是随着迭代级数减少的.

假设输入角度的位宽为24 bits,那么优化后的每一级角度运算加法器的运算位宽变化如表3所示.

表3 优化后的角度加法器运算位宽

Tab.3 Bit width of optimized angle adder

迭代级数i	角度加法器运算位宽/bits
1	24
2	23
3	22
4	21
…	…
22	2

角度加法器的位宽逐级缩减,只保留有效数据进行运算,去掉了无效位运算占用的硬件资源,有效避免了不必要的资源浪费.

2.2.2　向量模式

向量模式是通过一系列向量旋转逼近目标向量的工作方式,坐标 y 逐渐趋近于 0. 向量模式下的硬件资源优化就是利用坐标 y 的变化趋势,对运算过程中 x、y 的运算位宽进行缩减.

由第 i 级迭代公式(5)可以得出

$$\begin{cases} x_{i+1}^{'} = x_i^{'} - \delta_i y_i^{'} 2^{-i} = x_i^{'} + |y_i^{'} 2^{-i}|, \\ |y_{i+1}^{'}| = ||y_i^{'}| - x_i^{'} 2^{-i}|. \end{cases} \tag{17}$$

$x_i^{'}$ 是单调递增的. 经过数学变换后,x 和 y 的关系满足

$$|y_1^{'}| \leqslant x_1^{'}, 1 \geqslant x_1^{'} \geqslant 0. \tag{18}$$

代入式(17)后可以得出

$$|y_2^{'}| = ||y_1^{'}| - x_1^{'} 2^{-1}| \leqslant x_1^{'} 2^{-1} \leqslant 2^{-1}. \tag{19}$$

由于 $x_i^{'}$ 是单调递增的,可以推导出

$$\begin{cases} |y_3^{'}| = ||y_2^{'}| - x_2^{'} 2^{-2}| \leqslant x_2^{'} 2^{-2} \leqslant 2^{-2}, \\ |y_4^{'}| = ||y_3^{'}| - x_3^{'} 2^{-3}| \leqslant x_3^{'} 2^{-3} \leqslant 2^{-3}, \\ |y_5^{'}| = ||y_4^{'}| - x_4^{'} 2^{-4}| \leqslant x_4^{'} 2^{-4} \leqslant 2^{-4}, \\ \quad \vdots \\ |y_{i+1}^{'}| = ||y_i^{'}| - x_i^{'} 2^{-i}| \leqslant x_i^{'} 2^{-i} \leqslant 2^{-i}. \end{cases} \tag{20}$$

式(20)说明定点运算中 y 的有效位是不断减少的,对于 y 的加减运算位宽可以随着迭代级数逐级缩减. 假设输入数据位宽为 24 bits,那么优化后每一级 y 的加减运算位宽变化如表 4 所示.

表 4　优化后的 y 的加减运算位宽

Tab. 4　Bit width of addition and subtraction for y after optimization

迭代级数 i	y 的加减法运算位宽/bits
1	24
2	23
3	22
4	21
…	…
22	2

如表 4 所示,y 的加减运算位宽逐级缩减,只保留有效位进行运算,将无效位的运算占用的资源去掉了,有效避免了不必要的资源浪费.

由于 y 的有效位宽不断缩小,坐标 x 运算中的 $y_i^{'} 2^{-i}$ 项有效位宽也在不断的缩小. 假设输入数据位宽为 24 bits,优化后 x 的加减运算位宽变化如表 5 所示.

对向量(x_i, y_i)的优化,都是基于 CORDIC 算法的数据变化趋势以及定点运算的特点,不断去掉无效位以及无效运算,避免了不必要的运算资源的浪费,有效节省了硬件资源.

表 5　优化后的 x 的加减运算位宽

Tab. 5　Bit width of addition and subtraction for x after optimization

迭代级数 i	x 加减法运算位宽/bits
1	24
2	24
3	24
…	…
11	24
12	0
…	…
22	0

2.3　运算精度的提高

运算精度的提高主要通过数学变换后的数据格式化以及对运算路径位宽的扩展达到. 经过数据变换后,输入数据的有效位减少了,数据格式化就是通过移位的方式将减少的有效位补上,间接扩大了运算的数据位宽,提高运算精度. 运算路径的扩展一方面是为了提高运算精度,另一方面是为了保证某些特殊角的输出值.

2.3.1　数据格式化

数据格式化是将定点数的有效位进行移位,占据多余符号位的位置,将除符号位以外的位宽都变为有效位,从而达到增加运算精度.

定点数由符号位(S)、整数位(I)和小数位(F)三部分组成,如图 1 所示.

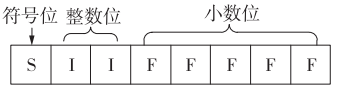


图 1　定点数的结构

Fig. 1　Structure of fixed-point

数据转换将[−π,π]范围内的角度转换到[−π/4,π/4]范围内,也就是说运算过程中角度的最大值变为原来的 1/4. 这意味着除符号位以外的第一个有效位移动到了第 4 位,也就是增加了 2 个多余的符号位,如图 2 所示.

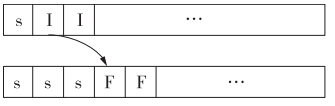


图 2　有效位移动示意图

Fig. 2　Valid bits moving diagram

数据格式化就是将有效位向左移动两位,保证有效位位宽不变. 同时也将用于旋转的特殊角 θ_i 也向左移动两位,保证运算的一致性.

同样在进行数学变换前坐标 x 的范围为[−1,

1]，进行数学变换后，迭代运算部分都是在0象限下进行的. 此时坐标 x 的范围缩小为[－0.707，0.707]，也就是说完全在小数范围内，整数位变为无效的符号位. 将有效位向左移动一位，保证有效位宽不变.

数据格式化后，有效数据的位宽进行了扩展，在一定程度上提高了运算的精度.

2.3.2 扩展数据路径的位宽

迭代次数以及数据路径的位宽直接影响着CORDIC算法的运算精度，Yu [9]于1992年提出了一种算法，该算法可以解决CORDIC算法迭代中基于总量化误差(OQE)的问题.

Yu提出的OQE由两种误差组成，分别是近似误差和舍入误差. 其中，近似误差是CORDIC旋转角度存在有限个基本角度量化所带来的量化误差，由迭代次数 n 和最大向量的模值 $v(0)$ 决定；舍入误差取决于实际运算中使用的有限精确度的代数运算，由数据路径的位宽 b 决定.

迭代次数 n 的增加和数据路径位宽 b 扩展都能提高运算的精度. 当迭代次数 n 较大时，继续增加迭代次数对近似误差的影响将很小. 而数据路径位宽每扩展1位，都将使舍入误差缩小为原来的1/2. 本设计在较大的迭代次数前提下，采用将数据路径位宽扩展5位的方式提高运算精度.

假设输入数据位宽为24 bits，则运算路径位宽扩展为29 bits. 图3是扩展数据路径位宽实现CORDIC算法的结构.

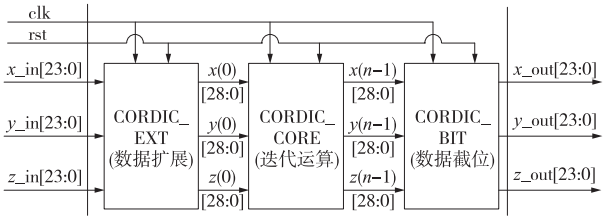


图3 扩展数据路径位宽的CORDIC算法

Fig. 3 CORDIC algorithm of data path bit width extended

由于CORDIC算法是一种逼近算法，运算结果与真实值之间始终存在一定误差，数据路径扩宽之后进行运算，最后再对数据进行截位舍入处理，能保证最终的输出数据更加接近真实值. 数据路径位宽扩展5位，一方面是为了提高运算精度，另一方面也是为了保证一些特殊角度值的运算，类似 $\sin(0)=0$ 的特殊角度运算.

数据格式化和数据路径位宽的扩展都是对有效运算位进行了扩展，对运算的精度有所提升.

3 优化CORDIC算法的实现

圆周系统下的旋转模式用于根据输入角度计算正弦和余弦值. 正余弦模块包括预处理、CORDIC迭代运算和后处理三个部分. 输入输出数据的位宽为24 bits，内部计算位宽为29 bits. 其中预处理部分实现输入寄存、数据转换和数据格式化，输入寄存可以根据具体应用选用；CORDIC迭代运算部分完成硬件资源缩减后的CORDIC旋转迭代运算；后处理部分根据预处理部分得到的象限信息，对输出数据进行象限恢复，并根据需要对输出数据进行截位处理，是预处理的一个逆向过程. 正余弦模块的硬件实现结构如图4所示.

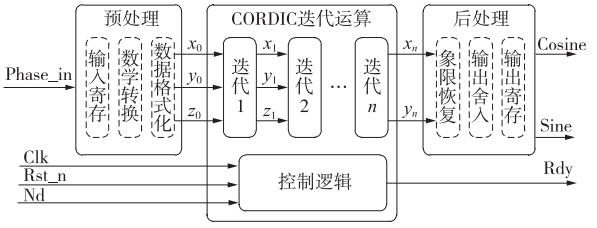


图4 正余弦运算模块的硬件结构

Fig. 4 Hardware architecture of sine and cosine module

圆周系统下的向量模式用于反正切函数的运算，根据输入的向量 (x,y) 计算出对应的反正切值. 反正切运算模块和正余弦模块一样，采用相同的流水线结构设计，只是具体细节和输入输出端口有所不同. 反正切运算模块的输入为向量的坐标值Xin和Yin，输出为角度值Phase.

4 仿真及综合结果分析

本设计在ISE12.4开发软件下，用VHDL硬件编程语言实现，使用Modelsim6.5进行仿真测试，并用Matlab验证仿真结果的正确性及相对误差.

在[－π，π]范围内均匀取点2 000个，用Matlab生成包含大批量测试数据的数据文件作为仿真的激励，来对优化的CORDIC算法进行仿真. 图5和图6分别是正余弦模块和反正切模块在Modelsim6.5下的仿真结果.

从图中可以看出，优化的正余弦运算模块和反正切模块的角度覆盖范围已经达到[－π，π]，完成了整个圆周的运算.

为了验证该优化算法的精度，在[0，π/2]范围内均匀取2 000个点，用Matlab生成激励数据. 然后

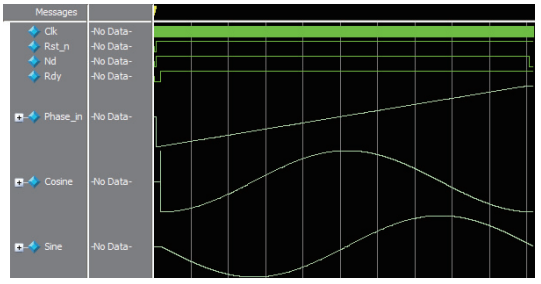


图 5　正余弦运算模块仿真结果

Fig. 5　Simulation result of sine and cosine module

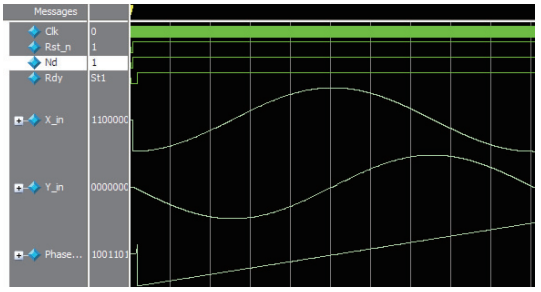


图 6　反正切运算模块仿真结果

Fig. 6　Simulation result of arctangent module

对正余弦运算模块和反正切运算模块分别进行仿真,并将仿真结果写入数据文件中. 最后通过 Matlab 将仿真数据读入,与 Matlab 自带的双精度浮点正余弦运算的结果比较.

图 7(a)为优化的正余弦运算相对误差,图 7(b)为由 Xilinx IP 核生成的正余弦运算相对误差. 由图 7(a)中可以看出,正余弦运算模块运算精度基本都在 $10^{-7}\sim10^{-6}$ 左右,只有输出数据非常小接近 0 的地方,相对误差较大. 比较图 7(a),7(b),可以看出优化的正余弦运算比由 Xilinx IP 核生成的正余弦运算的相对误差低. 例如在输入角度为 15°时,优化的正弦运算的相对误差为

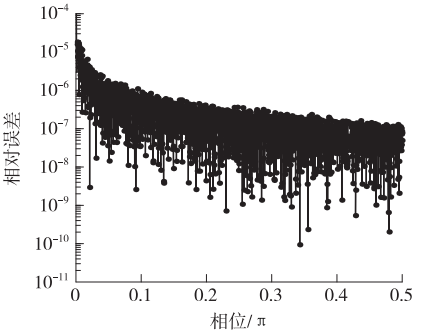
$$2.246\,297\,061\,886\,207\times10^{-7},$$

而由 Xilinx IP 核生成模块的正弦运算的相对误差为

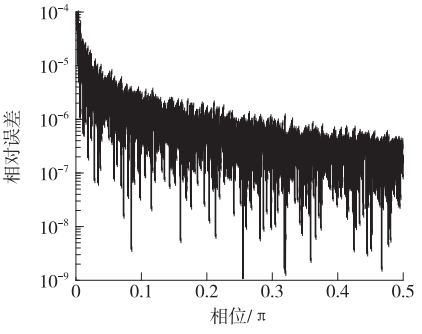
$$1.617\,727\,597\,982\,118\times10^{-6}.$$

图 8(a)为优化的反正切运算相对误差,图 8(b)为由 Xilinx IP 核生成的反正切运算相对误差. 可以看出,反正切运算模块的运算精度基本都在 10^{-7} 左右,只有输出数据非常小接近 0 的地方,相对误差较大. 对比图 8(a)8(b)可以看出,优化的反正切运算的精度比由 Xilinx IP 核生成的反正切运算模块的运算精度略高.

利用 Xilinx 公司的 XST 综合工具,选用 Virtex6 系列的 XC6VLX240T 芯片对优化的正余



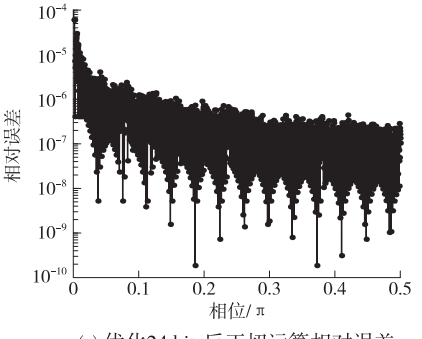
(a) 优化24 bits正余弦运算相对误差



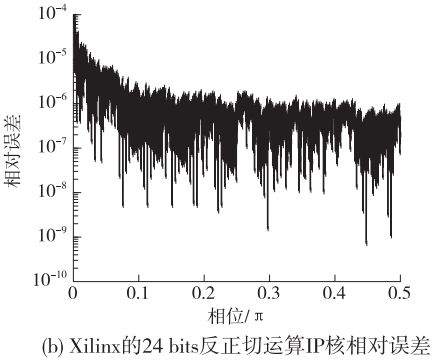
(b) Xilinx的24 bits正余弦运算IP核相对误差

图 7　优化的正余弦运算相对误差及 Xilinx IP 核的正余弦运算相对误差

Fig. 7　Relative errors of optimized sine and cosine module and that generated by Xilinx IP core



(a) 优化24 bits反正切运算相对误差



(b) Xilinx的24 bits反正切运算IP核相对误差

图 8　优化的反正切运算相对误差及 Xilinx IP 核的反正切运算相对误差

Fig. 8　Relative errors of optimized arctangent module and that generated by Xilinx IP core

弦运算模块、由 Xilinx IP 核生成的正余弦运算模

块、优化的反正切运算模块和由 Xilinx IP 核生成的反正切运算模块分别进行综合. 表 6～表 9 是上述 4 个模块的资源综合结果.

表 6 优化的正余弦模块资源综合结果

Tab. 6 Synthesis results of resources for optimized sine and cosine module

Logic Utilization	已用	可得	利用率/%
Number of Slice Registers	1 710	301 440	0
Number of Slice LUTs	1 838	150 720	1

表 7 Xilinx IP 核的正余弦模块资源综合结果

Tab. 7 Synthesis results of resources for sine and cosine module generated by Xilinx IP core

Logic Utilization	已用	可得	利用率/%
Number of Slice Registers	2 241	301 440	0
Number of Slice LUTs	2 301	150 720	1

表 8 优化的反正切正余弦模块资源综合结果

Tab. 8 Synthesis results of resources for optimized arctangent module

Logic Utilization	已用	可得	利用率/%
Number of Slice Registers	1 500	301 440	0
Number of Slice LUTs	1 552	150 720	1

表 9 Xilinx IP 核的反正切模块资源综合结果

Tab. 9 Synthesis results of resources for arctangent module generated by Xilinx IP core

Logic Utilization	已用	可得	利用率/%
Number of Slice Registers	2 174	301 440	0
Number of Slice LUTs	2 203	150 720	1

从综合报告得出 4 个模块的最高工作频率相同,均为 449.189 MHz. 对比表 6 和表 7 可知,在相同的工作频率下,优化的正余弦模块比由 Xilinx IP 核生成的正余弦模块节省了 23%的 Register 资源和 20%的 LUT 资源.

对比表 8 和表 9 可知,在相同的工作频率下,优化的反正切模块比由 Xilinx IP 核生成的反正切模块节省了 31%的 Register 资源和 29%的 LUT 资源.

5 结 论

本文通过对 CORDIC 算法进行分析,提出一系列优化措施以提高算法的精度并降低系统硬件资源消耗. 着重对正余弦运算的实现进行说明,并对正余弦运算的实现进行仿真、综合. 对仿真及综合结果的分析表明,本文对 CORDIC 算法的优化措施是切实有效的,能够真正提高算法精度并降低系统资源消耗.

参考文献:

[1] Volder J E. The CORDIC trigonometric computing technique [J]. IRE Transactions on Electronic Computers, 1959,8(3):330-334.

[2] 李滔,韩月秋. 基于流水线 CORDIC 算法的三角函数发生器[J]. 系统工程与电子技术,2000,22(4):85-87.
Li Tao, Han Yueqiu. Trigonometric function generator based on pipelined CORDIC [J]. Systems Engineering and Electronics, 2000,22(4):85-87. (in Chinese)

[3] 戚芳芳. CORDIC 算法的优化研究及其硬件实现[D]. 长沙:湖南大学,2012.
Qi Fangfang. The research of CORDIC algorithm improvement and its hardware implementation [D]. Changsha: Hunan University, 2012. (in Chinese)

[4] 孔德元. 针对正弦余弦计算的 CORDIC 算法优化及其 FPGA 实现[D]. 广州:中南大学,2008.
Kong Deyuan. Optimization of CORDIC algorithm for the calculation of sine and cosine and its implementation on FPGA [D]. Guangzhou: Central South University, 2008. (in Chinese)

[5] 张剑锋. 基于改进 CORDIC 算法的 DDFS 和 FFT 研究与实现[D]. 长沙:国防科技大学,2011.
Zhang Jianfeng. Research and implementation of DDFS and FFT based on the enhanced CORDIC [D]. Changsha: School of National University of Defense Technology, 2011. (in Chinese)

[6] Chen Dong, Chen He, Sun Xing, et al. Implementation of single-precision floating-point trigonometric functions with small area[C]// 2012 International Conference on Control Engineering and Communication Technology. Shanyang: IEEE, 2012:589-592.

[7] Yu Jiyang, Dan Huang, Pei Nan, et al. CORDIC-based design of matched filter weighted algorithm for pulse compression system[C]// 2012 IEEE 11th International Conference on Signal Processing. Beijing: IEEE, 2012: 1953-1956.

[8] Zhang Juntao, Ma Wenbo. Implementation of general CORDIC IP core based on FPGA[C]// 2011 IEEE International Conference on Computer Science and Automation Engineering. Shanghai: IEEE, 2011: 606-608.

[9] Hu H Y. The quantization effects of the CORDIC algorithm [J]. IEEE Trans. on Signal Processing, 1992,40:834-844.

(责任编辑:刘芳)