低时延-消耗的CORDIC算法及结构的研究

任小西 沈建龙
(湖南大学信息科学与工程学院 长沙 410082)

摘 要 CORDIC算法因便于通过硬件实现来计算各种超越函数而得到了广泛的应用。如何减少迭代次数并保持校正因子的计算与补偿的简单性是算法的难点,同时算法还需要扩展角度的范围。将常规的CORDIC算法分为前后两段,减少了迭代的次数,同时在硬件实现时使用移位操作代替查找表,减少了查找表所用的时间,这样做也有利于降低功耗。最后在Altera公司的Cyclone系列芯片EP4CGX22CF19C6上实现了该算法。实验结果表明:在同一数量级误差的基础上,该算法在结构上比常规算法节省了约34.84%的资源,在不同的工作频率上都少用了约6个时钟周期的时延,不同工作频率上系统的功耗最少也下降了约5.54%,并且工作频率越高,功耗下降越大。

关键词 CORDIC算法,低时延,资源,功耗,优化
中图法分类号 TP302.2 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.08.005

Reasearch on Low-latency and Low-consumption CORDIC Algorithm and Architecture

REN Xiao-xi SHEN Jian-long
(Department of Information Science and Engineering, Hunan University, Changsha 410082, China)

Abstract CORDIC algorithm has been widely used because it is easy to implement in hardware to calculate a variety of transcendental functions. How to reduce the number of iterations and maintain the calculation and the compensation of the correction factor is the difficulty of the algorithm, and it also needs to extend the range of rotation angle. In this paper, conventional CORDIC algorithm was divided into two steps to minimize the number of iterations. At the same time, the modified algorithm uses shift operations instead of the lookup table to reduce the time used by searching table, and also help to reduce power consumption. Finally, two algorithms were implemented in Altera Corporation Cyclone series chip EP4CGX22CF19C6. The experimental results show that compared to conventional algorithm, this algorithm saves 34.84% resources, and its delay is about six clock cycles less on various frequencies, and the power consumption on various frequencies declines about 5.54% at least, and the higher the frequency rises, the more the power consumption declines.

Keywords CORDIC algorithm, Low-latency, Resource, Power-consumption, Optimization

1 引言

坐标旋转数字计算(Coordinate Rotational Digital Computing,CORDIC)算法[1]是J. Volder等人于1959年在美国航空控制系统的设计中提出的,它是一种用于计算常用函数的循环迭代算法,其基本思想是通过一系列的只与运算基数有关的固定小角度的不断偏摆来逼近所要旋转的角度,从广义上来讲这是一种数值型计算逼近算法。为了扩展可计算函数的个数,1971年J. Walther提出了统一CORDIC算法[2],即把圆周坐标、线性坐标和双曲坐标统一到同一个CORDIC迭代方程中。

CORDIC算法因为能够将多种难以用硬件电路直接实现的复杂运算分解为简单的加减法和移位操作,所以很适合用数字电路来实现,因而其应用范围也就极其广泛,比如数控振荡器[3]、正余弦函数发生器[4]、数字频率合成器[5]等。

正因为CORDIC算法应用越来越广泛,提出了许多不同的方法来改进CORDIC算法,这些实现方式比常规CORDIC算法具有更少的迭代次数[6-11]。文献[6]利用贪婪搜索的方法减少了迭代次数,但却为校正模因子付出了额外空间和时间代价。文献[8]提出了一种高效的模因子补偿技术,但它却对延迟产生了不利的影响。文献[9]实现了一种低面积-时延的CORDIC结构,常数模因子只能在一定角度范围使用。文献[10]提出的scale-free算法同样也含有常数模因子,其适用角度范围更大,但因该算法只朝一个方向旋转,所以其每次旋转的角度会比常规CORDIC算法的小,因此迭代次数变多,必须想办法减少迭代次数。文献[11]提出一种增强的scale-free算法,其实就是将常规CORDIC算法和scale-free算法结合起来使用,该算法有一定的先进性。

本文提出的优化的CORDIC算法主要从4个方面进行了改进:1)查找表的移位实现;2)模校正因子的简化;3)减少迭代次数;4)旋转角度范围的扩展。运算速度和占用的芯片面积是衡量算法优劣的主要技术指标,本文最后利用FPGA实现流水线结构的CORDIC算法,通过与常规CORDIC算法的比较,证明本文提出的优化CORDIC算法提高了运算速

到稿日期:2013-10-26 返修日期:2014-01-04 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(1978-),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;沈建龙(1988-),男,硕士生,主要研究方向为可重构计算系统与多核计算系统,E-mail:shenjlong@163.com。

度，同时降低了系统所占用的面积，功耗也有所下降。

2 常规 CORDIC 算法

给定向量(x,y)，其旋转θ角后得到新向量(x',y')，如图1所示。

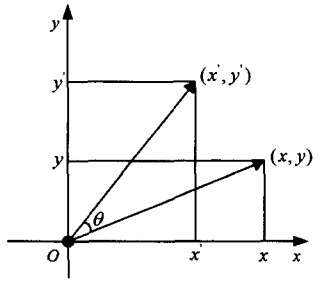

图1 CORDIC算法原理图

根据坐标变化规则，两向量之间有如下关系：

$$\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}=\cos\theta\begin{bmatrix}1 & -\tan\theta\\ \tan\theta & 1\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} \tag{1}$$

将旋转角θ分解为N个递减的小旋转角θ_i之和，即$\theta=\sum_{i=0}^{N-1}\delta_i\theta_i$，其中$\theta_i\geqslant 0$，$\theta_i$且当$\delta_i=1$时，第$i$次逆时针旋转；当$\delta_i=-1$时，第$i$次顺时针旋转。所以对于每一次小旋转，有如下旋转迭代公式：

$$\begin{bmatrix}x_{i+1}\\y_{i+1}\end{bmatrix}=\cos\theta_i\begin{bmatrix}1 & -\delta_i\tan\theta_i\\ \delta_i\tan\theta_i & 1\end{bmatrix}\begin{bmatrix}x_i\\y_i\end{bmatrix},i=0,1,2,\cdots,N-1 \tag{2}$$

为了便于硬件实现，令$\theta_i=\arctan(2^{-i})$，即$\tan\theta_i=2^{-i}$，这时有如下关系：

$$\cos\theta_i=\frac{1}{\sqrt{1+\tan^2\theta_i}}=\frac{1}{\sqrt{1+2^{-2i}}} \tag{3}$$

旋转迭代公式(2)可以修改为：

$$\begin{bmatrix}x_{i+1}\\y_{i+1}\end{bmatrix}=\cos\theta_i\begin{bmatrix}1 & -\delta_i 2^{-i}\\ \delta_i 2^{-i} & 1\end{bmatrix}\begin{bmatrix}x_i\\y_i\end{bmatrix},\quad i=0,1,2,\cdots,N-1 \tag{4}$$

将式(4)中的$\cos\theta_i$项去除，得到如下伪旋转迭代公式(5)：

$$\begin{bmatrix}x_{i+1}\\y_{i+1}\end{bmatrix}=\begin{bmatrix}1 & -\delta_i 2^{-i}\\ \delta_i 2^{-i} & 1\end{bmatrix}\begin{bmatrix}x_i\\y_i\end{bmatrix},i=0,1,2,\cdots,N-1 \tag{5}$$

通过这一步去除$\cos\theta_i$项，旋转的角度还是正确的，但是x_{i+1}和y_{i+1}的值都增加$1/\cos\theta_i$倍，所以旋转后的向量模值变大，如图2所示。注意理论上并不能通过数学方法去除$\cos\theta_i$项，然而我们发现去除$\cos\theta_i$项可以简化计算。

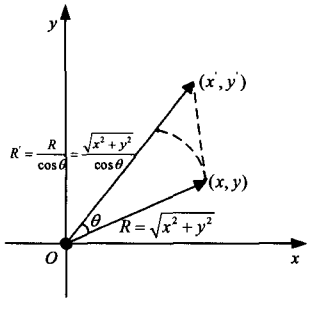

图2 模值变大

为了追踪已经旋转的角度，引入第三个方程，称为角度累加器：

$$\begin{cases}z_{i+1}=z_i-\delta_i\theta_i=z_i-\delta_i\arctan(2^{-i})\\ \delta_i=sign(z_i)\end{cases},i=0,1,2,\cdots,N-1 \tag{6}$$

z_i表示第i次已经旋转的角度，z_{i+1}表示经过第i次旋转后剩余未旋转的角度。其中$\arctan(2^{-i})$可以预先求出，保存在ROM中用于查询。

综上分析，最终旋转迭代公式如下：

$$\begin{cases}x_{i+1}=x_i-\delta_i 2^{-i}y_i\\ y_{i+1}=y_i+\delta_i 2^{-i}x_i\\ z_{i+1}=z_i-\delta_i\arctan(2^{-i})\\ \delta_i=sign(z_i)\end{cases},i=0,1,2,\cdots N-1 \tag{7}$$

正如前面所述，去除了$\cos\theta_i$项，输出x_{i+1}和y_{i+1}的值都增加$1/\cos\theta_i$倍，所以旋转后的向量模值变大。给定向量(x,y)，根据式(4)，经过N步旋转之后得到新向量(x_N,y_N)，如式(8)所示：

$$\begin{aligned}\begin{bmatrix}x_N\\y_N\end{bmatrix}&=\left(\prod_{i=0}^{N-1}\cos\theta_i\begin{bmatrix}1 & -\delta_i 2^{-i}\\ \delta_i 2^{-i} & 1\end{bmatrix}\right)\begin{bmatrix}x_0\\y_0\end{bmatrix}\\&=K_N\cdot\left(\prod_{i=0}^{N-1}\begin{bmatrix}1 & -\delta_i 2^{-i}\\ \delta_i 2^{-i} & 1\end{bmatrix}\right)\begin{bmatrix}x_0\\y_0\end{bmatrix}\end{aligned} \tag{8}$$

其中，K_N称为为模校正因子，其是伪旋转不可避免的副产物，表达式如式(9)所示：

$$K_N=\prod_{i=0}^{N-1}\cos\theta_i=\prod_{i=0}^{N-1}\frac{1}{\sqrt{1+2^{-2i}}} \tag{9}$$

当$N\to\infty$时，有$K_N\to 0.607253$。当迭代次数N的值确定时，K_N的值也可以确定，所以K_N可以看作一个常量，我们就可以在设计系统时提前计算K_N的值，这样就可以正常地使用最终旋转迭代公式(7)了。

3 改进 CORDIC 算法

3.1 查找表的移位实现

CORDIC算法在每级迭代时都需要查找ROM中预存的值$\arctan(2^{-i})$，查找表会占用大量的ROM资源($2n\times n$bit)，即随着N的增加，ROM表的容量呈现指数式的增长[12]，占用更多的硬件资源，并且每次迭代都访问ROM表，降低了系统的速度[13]，很容易对流水线形成“瓶颈”。为了节省硬件资源，减少系统面积，需要想办法避免查找表。

先以每次旋转的角度$\theta_i=\arctan(2^{-i})$为研究出发点，用麦克劳林级数展开：

$$\arctan(2^{-i})=2^{-i}-\frac{1}{3}\cdot 2^{-3i}+\frac{1}{5}\cdot 2^{-5i}-\frac{1}{7}\cdot 2^{-7i}+\frac{1}{9}\cdot 2^{-9i}-\frac{1}{11}\cdot 2^{-11i}+\cdots \tag{10}$$

从上式可以看到，$\arctan(2^{-i})$和2^{-i}之间的差别随着i的增加而呈指数级减少，设想若从m级迭代开始用2^{-m}代替$\arctan(2^{-i})$，$i\geqslant m$所产生的误差对最终结果没有影响，则此时查ROM表的操作便可以用移位运算取代，这样便可以减少ROM资源的占用。理论的关键是用2^{-m}代替$\arctan(2^{-i})$，$i\geqslant m$时所产生的误差可以忽略不计。对于位宽为N位的CORDIC算法，系统的精度$\varepsilon=2^{-N+1}$，所以：

$$|2^{-i}-\arctan(2^{-i})|\leqslant 2^{-N+1} \tag{11}$$

将式(10)代入式(11)，得到式(12)：

$$i \geqslant m = \left\lceil \frac{1}{3} \cdot (N-1-\log_2 3) \right\rceil,$$

$\lceil * \rceil$表示取 $*$ 的最大整数 (12)

若 $N=16$,可以求出 $m=5$;若 $N=32$,可以求出 $m=11$。

可见当迭代到第 m 级流水线时,即当 $i \geqslant m$ 时,若用 2^{-m} 代替 $\arctan(2^{-i})$,$i \geqslant m$,不再需要从 ROM 表中查找 $\arctan(2^{-i})$值,节省了访问 ROM 的时间,提高了计算速度,还能减少大概 2/3 的 ROM 资源。

3.2 模校正因子的简化

CORDIC 算法的向量旋转并非完美,因为旋转的过程会使向量的模变化,这就需要对模进行校正,模校正因子会给每一步的迭代带来乘法运算,这就与 CORDIC 算法的初衷相违背了,乘法运算增加了硬件的实现难度,降低了数据的处理速度,同样需要想办法化简模校正因子。

将 $\cos\theta_i$ 用麦克劳林级数展开:

$$\cos\theta_i = \frac{1}{\sqrt{1+2^{-2i}}} = 1-2^{-2i-1}+3\cdot 2^{-4i-3}-\cdots \quad (13)$$

从上式可以看到,$\cos\theta_i$ 和 $1-2^{-2i-1}$之间的差别随着 i 的增加而呈指数级减少,设想若从 l 级迭代开始用 $\cos\theta_i$ 代替 $1-2^{-2i-1}$,$i \geqslant l$,理论的关键是使用 $\cos\theta_i$ 代替 $1-2^{-2i-1}$,$i \geqslant l$ 时所产生的误差可以忽略不计。对于位宽为 N 位的 CORDIC 算法,系统的精度 $\varepsilon = 2^{-N+1}$,所以:

$$|\cos\theta_i - (1-2^{-2i-1})| \leqslant 2^{-N+1} \quad (14)$$

将式(13)代入式(14),得到式(15):

$$i \geqslant l = \left\lceil \frac{1}{4} \cdot (N-4+\log_2 3) \right\rceil, \lceil * \rceil \text{表示取} * \text{的最大整数} \quad (15)$$

若 $N=16$,可以求出 $l=4$;若 $N=32$,可以求出 $l=8$。

所以当 $N=16$,$i \geqslant l=4$ 时或 $N=32$,$i \geqslant l=8$ 时,式(4)可以改写为:

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = (1-2^{-2i-1}) \begin{bmatrix} 1 & -\delta_i 2^{-i} \\ \delta_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}, \quad i=0,1,2,\cdots,N-1 \quad (16)$$

即当 $N=16$,$i \geqslant l=4$ 时或 $N=32$,$i \geqslant l=8$ 时,有 $1-2^{-2i-1} \approx 1$,即 $K_N = \prod_{i=m}^{N-1} \cos\theta_i = \prod_{i=m}^{N-1} (1-2^{-2i-1}) = 1$,这种情况下就简化了模校正因子。

为了简化电路的设计,设查找表的移位和模校正因子的简化同时实现,这只有 l 在 $i \geqslant \max(m,l)$的情况下才满足,即当 $N=16$,$i \geqslant 5$ 或 $N=32$,$i \geqslant 11$ 时。

3.3 减少迭代次数

在 $N=16$,$i \geqslant 5$ 或 $N=32$,$i \geqslant 11$ 时,我们还可以进一步地减少不必要的迭代。因为在这样的条件下,每一步旋转的角度都很小,所以没有必要按部就班地按照常规 CORDIC 算法的步骤进行旋转,若规定前后两次旋转至少有一次不用旋转,即 δ_i 和 δ_{i+1}中至少有一个为 0,表示不旋转,则可以利用式(5)、式(11)和式(14)化简式(17):

$$\begin{cases} x_{i+2} = x_{i+1} - \delta_{i+1} 2^{-i-1} y_{i+1} = x_i - (\delta_i 2^{-i} + \delta_{i+1} 2^{-i-1}) y_i \\ y_{i+2} = y_{i+1} + \delta_{i+1} 2^{-i-1} x_{i+1} = y_i + (\delta_i 2^{-i} + \delta_{i+1} 2^{-i-1}) x_i \\ z_{i+2} = z_{i+1} - \delta_{i+1} \arctan(2^{-i-1}) = z_i - \delta_i 2^{-i} - \delta_{i+1} 2^{-i-1} \end{cases}, \quad i \geqslant \max(m,l) \quad (17)$$

通过上式可以发现只要选择适当的 δ_i 和 δ_{i+1},两步迭代就可以合并成一步,这样便加速了未旋转角度趋于 0,减少流水线的级数,提高运算速度。

在进行第 i 次迭代后,z_i 和 θ_i 二进制数的高 $N-i-1$ 位之差为 0,所以可以根据 $z_i^{N-i-1} z_i^{N-i-2}$ 位取不同的值,然后按照式(17)选择合适的 δ_i 和 δ_{i+1},使得其能够保证(z_{i+2}^{N-i-1},z_{i+2}^{N-i-2})=(0,0)即可到达计算要求。按照这样的规则总结出 δ_i 和 δ_{i+1}取值,如表 1 所列。

表 1 取值表

z_i^{N-1}	$(z_i^{N-i-1}, z_i^{N-i-2})$	(δ_i, δ_{i+1})
0	(0,0)	(0,0)
	(0,1)或(1,0)	(0,1)
	(1,1)	(1,0)
1	(0,0)	(−1,0)
	(0,1)或(1,0)	(0,−1)
	(1,1)	(0,0)

对于 $N=16$ 的 CORDIC 算法,先由式(7)迭代 5 次,模校正因子 $K_N = \prod_{i=0}^{4} \cos\theta_i = \prod_{i=0}^{4} \frac{1}{\sqrt{1+2^{-2i}}} = 0.607648$,后面的 11 级使用式(17)迭代 6 次,模校正因 $K_N=1$,不必进行修正,因此整个算法只需要 11 次迭代和 5 个 ROM 存储单元,减少了 5 级流水线和 11 个 ROM 存储单元,从而降低了 CORDIC 的硬件消耗,减少了 ROM 访问次数,提高了系统速度,也有利于降低系统功耗。

对于 $N=32$ 的 CORDIC 算法,先由式(7)迭代 11 次,模校正因子 $K_N = \prod_{i=0}^{10} \cos\theta_i = \prod_{i=0}^{10} \frac{1}{\sqrt{1+2^{-2i}}} = 0.607259$,后面的 21 级使用式(17)迭代 11 次,模校正因子 $K_N=1$,不必进行修正,因此整个算法只需要 22 次迭代和 11 个 ROM 存储单元,减少了 10 级流水线和 21 个 ROM 存储单元,降低了 CORDIC 的硬件消耗,减少了 ROM 访问次数,提高了系统速度,降低了系统功耗。

3.4 旋转角度的范围扩展

CORDIC 算法的旋转角度只能覆盖[−99.88°,99.88°]范围,而实际上我们需要能够覆盖整个周期的算法。

本文采用分象限法扩大旋转角度范围,所谓分象限法主要是利用三角函数的对称性,将整个周期内的角度全部转换到第一象限。根据转换规则,前置模块保存相位信息,经过 CORDIC 算法模块后,后置模块再根据相位信息恢复原相位,转换规则如表 2 所列。

表 2 相位转换规则

输入角度	转换角度	x_N	y_N
$0° \leqslant z_0 \leqslant 90°$	z_0	x_{15}(N=16)或 x_{31}(N=32)	y_{15}(N=16)或 y_{31}(N=32)
$90° \leqslant z_0 \leqslant 180°$	$z_0-90°$	$-y_{15}$(N=16)或 $-y_{31}$(N=32)	x_{15}(N=16)或 x_{31}(N=32)
$180° \leqslant z_0 \leqslant 270°$	$z_0-180°$	$-x_{15}$(N=16)或 $-x_{31}$(N=32)	$-y_{15}$(N=16)或 $-y_{31}$(N=32)
$270° \leqslant z_0 \leqslant 360°$	$z_0-270°$	y_{15}(N=16)或 y_{31}(N=32)	$-x_{15}$(N=16)或 $-x_{31}$(N=32)

4 系统设计及实现

基于 FPGA 优化设计的 CORDIC 算法的整体系统结构框图如图 3 所示。

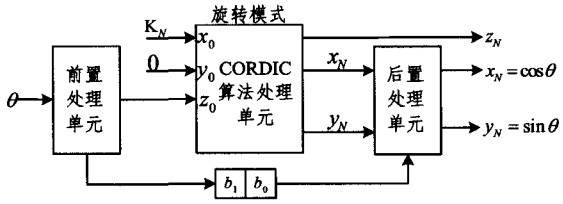

图 3　整体结构框图

本文所有结构使用 Verilog HDL 实现，用 Quartus Ⅱ 编译综合，用 Mentor Graphics Modelsim SE 仿真。

5　数据比较分析

5.1　误差分析

编写了 Testbench 来测试不同输入值下两种算法的计算值和误差，结果如表 3 所列。

表 3　误差分析

角度		标准值	常规 CORDIC(N=16)		改进 CORDIC(N=16)	
			值	绝对误差	值	绝对误差
45°	cos	0.7071067812	0.7102966309	3.19×10^{-3}	0.6992492676	7.86×10^{-3}
	sin	0.7071067812	0.7039184570	3.12×10^{-3}	0.7141723633	7.07×10^{-3}
160°	cos	−0.9396926208	−0.9380187988	1.67×10^{-3}	−0.9333300781	6.36×10^{-3}
	sin	0.3420201433	0.3464660645	4.44×10^{-3}	0.3498298340	7.81×10^{-3}
230°	cos	−0.6427876097	−0.6461486816	3.36×10^{-3}	−0.6355076097	7.28×10^{-3}
	sin	−0.7660444431	−0.7632446289	2.80×10^{-3}	−0.7619348145	4.11×10^{-3}
300°	cos	0.5000000000	0.4973754883	2.62×10^{-3}	0.4920065918	7.00×10^{-3}
	sin	−0.8660254038	−0.8674316406	1.41×10^{-3}	−0.8721008301	6.08×10^{-3}

分析表 3 可知，在误差方面，虽然改进的 CORDIC 算法误差比常规 CORDIC 算法大一些，但是二者都在同一个数量级上，所以改进的 CORDIC 算法舍入误差和近似误差并没有改变计算精度。

5.2　面积分析

根据 CORDIC 算法以及改进 CORDIC 算法的分析，理论上得出算法实现所消耗的资源，如表 4 所列。

表 4　面积需求

算法类型	流水线结构(N=16)			
	选择器	加减法单元	移位单元	ROM 单元
常规 CORDIC	45	45	30	16
改进 CORDIC	33	33	22	5

利用 Quartus Ⅱ，选用 EP4CGX22CF19C6 芯片，进行综合后得到的报告是：常规 CORDIC 算法使用了 1019 个逻辑单元，而改进的 CORDIC 算法使用了 664 个逻辑单元，改进 CORDIC 算法比常规 CORDIC 算法资源消耗降低了 34.84%。

5.3　最高工作频率与时延分析

使用 Quartus 进行综合后，得到电路的最高工作频率，这个指标取决于流水线结构中耗时最长的那一段流水的时延，如表 5 所列。

表 5　最高工作频率分析

算法类型	流水线结构(N=16)
	F_{max}
常规 CORDIC	199.16MHz
改进 CORDIC	224.47MHz
上升率	11.28%

从表中可以看到，在最高工作频率方面，改进的 CORDIC 算法比常规 CORDIC 算法提高了约 11.28%，表明改进的算法优化较好。

使用 Modelsim 进行仿真，得到不同工作频率下两种算法的时延，如表 6 所列。

从表中可以看到，改进的 CORDIC 算法在不同的工作频率上时延都有所下降，并且相较于常规 CORDIC 算法，在各个工作频率上少用了大约 6 个时钟周期的时延，运算速度提高了。

表 6　时延分析

算法类型	流水线结构(N=16)					
	100MHz	200MHz	300MHz	400MHz	600MHz	800MHz
常规 CORDIC	190ns	95ns	66.2ns	47.5ns	32.88ns	23.75ns
改进 CORDIC	130ns	65ns	46.122ns	32.5ns	22.884ns	16.25ns
少用时钟周期	6	6	6	6	6	6

5.4　功耗分析

使用 PowerPlay 功耗分析套件进行分析，得到在不同频率下两种算法的功率，如表 7 所列。

表 7　功率分析

算法类型	流水线结构(N=16)					
	100MHz	200MHz	300MHz	400MHz	600MHz	800MHz
常规 CORDIC	106.77mW	119.50mW	131.97mW	144.85mW	169.98mW	195.48mW
改进 CORDIC	100.85mW	107.83mW	114.57mW	121.65mW	135.26mW	140.20mW
下降率	5.54%	9.77%	13.19%	16.02%	20.43%	28.28%

从表中可以看到，改进的 CORDIC 算法在不同的工作频率上功率都有所下降，并且工作频率越高，功率下降越大。

结束语　本文设计了一个改进的 CORDIC 算法，针对常规 CORDIC 算法的局限性，它成功解决了速度、面积、精度之间的制约关系，打破了角度覆盖范围的限制。通过压缩 ROM 容量，节省了访问 ROM 的时间，提高了运算速度，同时还能够减少系统的面积；通过减少模校正因子，简化了硬件的实现难度，减少了系统的面积；通过去掉一些不必要的迭代，减少了流水线级数，提高了系统的运算速度。新的结构经过综合、仿真后，在一定精度范围内，相较于最初的结构资源消耗降低了 34.84%，最高工作频率提高了约 11.28%，时延少用了 6 个时钟周期，并且工作频率越高，功率下降越大。

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] Walther J S. A unified algorithm for elementary functions[J]. AFIPS Spring Joint Computer Conference, 1971, 38: 379-385

[3] Lei Zhi-hua. The design of NCO based on CORDIC algorithm and implementation in FPGA[C]// 2011 International Conference on Electronics, Communications and Control (IECECC). 2011: 2902-2905

[4] Considine V. CORDIC Trigonometric Function Generator for DSP[C]// International Conference on Acoustics, Speech, and Signal Processing, ICASSP-89. 1989, 4: 2381-2384

[5] Wang Xiao-chu. High-Precision Design of DDS Based on FPGA [C]// 2012 Third Global Congress on Intelligent Systems (GCIS). 2012: 386-389

[6] Wu C S, Wu A Y, Lin C H. A high-performance/low-latency vector rotational CORDIC architecture based on extended elementary angle set and trellis-based searching schemes[J]. IEEE Trans. Circuits Syst., 2003, 50(9): 589-601

[7] 李美俊，李光明. 基于嵌入式的 CORDIC 算法的改进及实现[J]. 微电子学与计算机，2012，55(2)：33-34

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

[9] Vachhani L, Sridharan K, Meher P K. Efficient CORDIC algorithmsand architectures for low area and high throughput implementation[J]. IEEE Trans. Circuit Syst., 2009, 56(1): 61-65

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

[11] Jaime F J, Sanchez M A, Hormigo J, et al. Enhanced scaling-free CORDIC[J]. IEEE Trans. Circuits Syst., 2010, 57(7): 1654-1662

[12] 常柯阳，曾岳南. CORDIC 算法在正余弦函数中的应用及其 FPGA 实现[J]. 计算机工程与应用，2013，34(7)：143-144

[13] Baese U M. Digital signal processing with field programmable gate arrays[M]. 刘凌，胡永生，译. 北京：清华大学出版社，2003

（上接第 6 页）

[58] Zheng W, Yong J H, Paul J C. Simulation of bubbles[J]. Graphical Models, 2009, 71(6): 229-239

[59] 朱红斌，刘学慧，柳有权，等. 基于 Lattice Boltzmann 模型的液-液混合流模拟[J]. 计算机学报，2006，29(12)：2071-2079

[60] 武小龙，吴恩华. 气泡的生成和多种流体的模拟[J]. 计算机辅助设计与图形学学报，2010(9)：1463-1467

[61] Orthmann J, Kolb A. Temporal Blending for Adaptive SPH [J]. Computer Graphics Forum, 2012, 31(8): 2436-2449

[62] Busaryev O, Dey T K, Wang H, et al. Animating bubble interactions in a liquid foam [J]. ACM Transactions on Graphics (TOG), 2012, 31(4): 63

[63] Reagin D, Lake A. Real-Time Deep Ocean Simulation on Multi-Threaded Architectures[EB]. https://software.intel.com/en-us/articles

[64] Chiu P, Lee L, Sheu T W. A dispersion-relation-preserving algorithm for a nonlinear shallow-water wave equation[J]. Journal of Computational Physics, 2009, 228(21): 8034-8052

[65] 李永进，金一丞，任鸿翔，等. 基于物理模型的近岸海浪建模与实时绘制[J]. 中国图象图形学报，2010(3)：518-523

[66] Ihmsen M, Akinci N, Becker M, et al. A Parallel SPH Implementation on Multi-Core CPUs [J]. Computer Graphics Forum, 2011, 30(1): 99-112

[67] Buck I, Purcell T. A toolkit for computation on GPUs[M]// GPU Gems. Addison-Wesley Professional 2004: 621-636

[68] Bolz J, Farmer I, Grinspun E, et al. Sparse matrix solvers on the GPU: conjugate gradients and multigrid[J]. ACM Transactions on Graphics (TOG), 2003, 22(3): 917-924

[69] Moreland K, Angel E. The FFT on a GPU[C]// Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware. 2003: 112-119

[70] Sumanaweera T, Liu D. Medical image reconstruction with the FFT[J]. GPU Gems, 2005(2): 765-784

[71] Harris M. Fast fluid dynamics simulation on the GPU[J]. GPU gems, 2004(1): 637-665

[72] 柳有权，刘学慧，吴恩华. 基于 GPU 带有复杂边界的三维实时流体模拟[J]. 软件学报，2006，17(3)：568-576

[73] Amada T, Imura M, Yasumuro Y, et al. Particle-based fluid simulation on GPU [C]// ACM Workshop on General-Purpose Computing on Graphics Processors and SIGGRAPH. 2004

[74] Kolb A, Cuntz N. Dynamic particle coupling for GPU-based fluid simulation[C]// Proc. Symposium on Simulation Technique. 2005: 722-727

[75] Harada T, Koshizuka S, Kawaguchi Y. Smoothed particle hydrodynamics on GPUs [C]// Computer Graphics International. 2007: 63-70

[76] Hérault A, Bilotta G, Dalrymple R A. SPH on GPU with CUDA [J]. Journal of Hydraulic Research, 2010, 48(S1): 74-79

[77] 温婵娟，欧嘉蔚，贾金原. GPU 通用计算平台上的 SPH 流体模拟[J]. 计算机辅助设计与图形学学报，2010(3)：406-411

[78] 陈曦，王章野，何戡，等. GPU 中的流体场景实时模拟算法[J]. 计算机辅助设计与图形学学报，2010(3)：396-405

[79] Liu Y, Shi K, Deng H, et al. A multi-GPU based semi-Lagrangian fluid solver[C]// Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry. 2011: 321-326

[80] Losasso F, Gibou F, Fedkiw R. Simulating water and smoke with an octree data structure[J]. ACM Transactions on Graphics (TOG), 2004, 23(3): 457-462

[81] Feldman B E, O'Brien J F, Klingner B M. Animating gases with hybrid meshes [J]. ACM Transactions on Graphics (TOG), 2005, 24(3): 904-909

[82] Green S. Cuda particles[M]. NVidia Whitepaper, 2008