双模式 CORDIC 算法的 FPGA 实现

王睐来　杨春玲
（哈尔滨工业大学电气工程及自动化学院，哈尔滨　150001）

摘要：CORDIC 算法将复杂的算术运算转化为简单的加法和移位操作，然后逐次逼近结果。这种方法很好的兼顾了精度、速度和硬件复杂度，它与 VLSI 技术的结合对 DSP 算法的硬件实现具有极大的意义，因而在数字信号处理领域得到了广泛应用。本文首先简要介绍了 CORDIC 算法的原理，然后详细描述了双模式（旋转/向量）CORDIC 算法的预处理和后处理，并且基于 FPGA 实现了流水线双模 CORDIC 算法。

关键词：CORDIC 算法；FPGA

Implementation of dual-mode CORDIC based on FPGA

WANG Jian-lai　YANG Chun-ling
(Electrical Engineering, Harbin Institute of Technology, Harbin, 150001)

Abstract: By converting complex arithmetic into simple operations such as adding and shift then gradually approach the exact result, CORDIC algorithm keeps balance between precision, speed and hardware complexity, and whose combination with VLSI performs a great role in the hardware implementation of DSP algorithm, enhance CORDIC has been widely used in the field of digital signal processing. We studied the principle of CORDIC algorithm, and described the pre and post process of dual mode CORDIC algorithm, then the pipelined dual-mode CORDIC algorithm is implemented based on FPGA.

Keywords: CORDIC; FPGA

引言

CORDIC（Coordinate Rotational Digital Computer，坐标旋转计算机）[1,2]算法是Volder于 1959 年在美国航空控制系统的设计中提出来的，它是一种用于计算运算函数的循环迭代算法。其基本思想是用一系列与运算基数相关的角度的不断偏摆，从而逼近所需旋转的角度。从广义上讲它是一个数值计算逼近的方法。这些固定的角度与计算基数有关，运算只有移位和加减。可通过该算法不同的实现形式（如圆周模式、双曲线模式、线性模式等）来计算包括乘、除、平方根、正旋、余弦、反正切、向量旋转（即复数乘法）以及指数运算等[3]。1971 年，J.S Walther[4]提出了统一的CORDIC算法形式，把圆周旋转、双曲旋转和直线旋转统一到同一个CORDIC迭代方程中，为统一硬件实现多功能运算奠定了基础。

传统上计算三角函数和其它一些硬件不易实现的函数，一般使用查表法、多项式展开或近似的方法。这些方法不能兼顾速度、精度、简单性等方面的要求。CORDIC 算法正是为解决这种问题而产生的。它从算法本身入手，将复杂的算法分解成一些在硬件中容易实现的基本算法，如加法、移位等，从而使得这些算法在硬件上可以得到较好的实现。由于该算法是一种规则化的算法，它满足了硬件对算法的模块化、规则化的要求，因此 CORDIC 算法可以充分发挥硬件的优势，利用硬件的资源，从而实现硬件与算法相结合的一种优化方案。

正是由于上述各种原因，CORDIC 算法的原始思想一经提出，就受到人们的普遍关注。40 年来人们不断的对其进行探索，并提出各种改进算法和优化方案，以适应各种不同的需求。CORDIC 算法已应用在包括适应性滤波器、FFT、DCT、解调器和神经网络等诸多领域中。

1　CORDIC 算法原理

本文以CORDIC的圆周模式为例阐述CORDIC算法的两种工作模式：旋转模式和向量模式[5]。

1.1 CORDIC 旋转模式

假设直角坐标系内有一个向量 $a(X_a,Y_a)$，逆时针旋转 θ 角度后得到另一个向量 $b(X_b,Y_b)$，如图 1 所示。

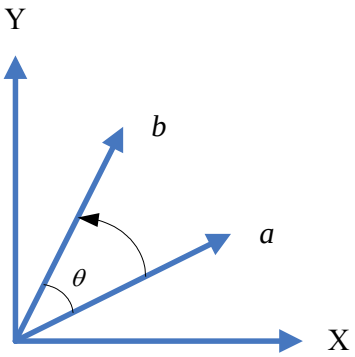

图 1 旋转模式单次旋转图

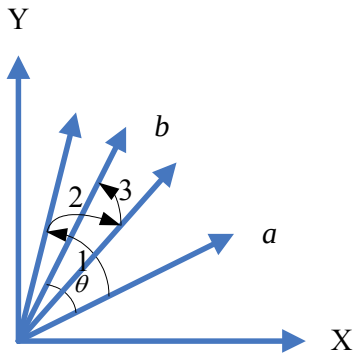

图 2 旋转模式多次旋转图

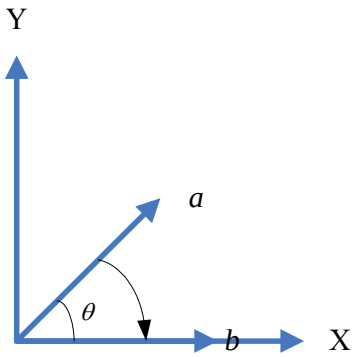

图 3 向量模式旋转图

这个过程可以用矩阵形式表示如下：

$$\begin{pmatrix} X_b \\ Y_b \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} X_a \\ Y_a \end{pmatrix} \tag{1}$$

把 $\cos\theta$ 提出来，式(1)可以重新表示为：

$$\begin{pmatrix} X_b \\ Y_b \end{pmatrix} = \cos\theta \begin{pmatrix} 1 & -\tan\theta \\ \tan\theta & 1 \end{pmatrix} \begin{pmatrix} X_a \\ Y_a \end{pmatrix} \tag{2}$$

如果向量 $a(X_a,Y_a)$ 经过 n 次旋转才到达向量 $b(X_b,Y_b)$，如图 2 所示：

其中第 i 次旋转的角度为 θ_i，那么第 i 次旋转的表达式为：

$$\begin{pmatrix} X_{i+1} \\ Y_{i+1} \end{pmatrix} = \cos\theta_i \begin{pmatrix} 1 & -\tan\theta_i \\ \tan\theta_i & 1 \end{pmatrix} \begin{pmatrix} X_i \\ Y_i \end{pmatrix} \tag{3}$$

这里取 $\tan\theta_i = S_i \frac{1}{2^i}$，即 $\theta_i = S_i \arctan(\frac{1}{2^i})$，旋转的角度总和 $\theta = \sum_{i=0}^{\infty} \theta_i$ [6]，这里 $S_i = \{-1;+1\}$，当 $S_i = +1$ 时表示向量是逆时针旋转，当 $S_i = -1$ 时表示向量是顺时针旋转。这时式(3)变为：

$$\begin{pmatrix} X_{i+1} \\ Y_{i+1} \end{pmatrix} = \cos\theta_i \begin{pmatrix} 1 & -S_i 2^{-i} \\ S_i 2^{-i} & 1 \end{pmatrix} \begin{pmatrix} X_i \\ Y_i \end{pmatrix} \tag{4}$$

式(4)中的 $\cos\theta_i = \cos(S_i \arctan(\frac{1}{2^i})) = \cos(\arctan(\frac{1}{2^i}))$。随着旋转次数的增加，该式收敛为一个常数 k，

$$k = \prod_{i=0}^{\infty} \cos(\arctan(\frac{1}{2^i})) \approx 0.607253 \tag{5}$$

上式中 k 是一个常数增益，暂时不考虑这个增益因子，式(4)可以写为：

$$\begin{pmatrix} X_{i+1} \\ Y_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -S_i 2^{-i} \\ S_i 2^{-i} & 1 \end{pmatrix} \begin{pmatrix} X_i \\ Y_i \end{pmatrix} \tag{6}$$

式(6)就是 CORDIC 的迭代式，它只需要通过移位和相加就可以完成矢量的旋转。

设 Z_0 是 a，b 两向量的夹角，要使 a 旋转到 b，则 S_i 的符号由第 i 次旋转时的角度 Z_i 决定，$Z_i = Z_0 - \sum_{n=0}^{i-1} \theta_n$，它们的关系如下：$Z_i < 0$ 时，$S_i = -1$；$Z_i \geq 0$ 时，$S_i = +1$（特殊情况下，$Z_0 < 0$ 时，$S_0 = -1$；$Z_0 \geq 0$ 时，$S_0 = +1$）。

向量 a 向向量 b 逼近的精度由迭代的次数决定，迭代的次数越多，逼近的精度就越高。

迭代 n 次（$n \to \infty$）得到的最终结果为：

$$\begin{cases} X_n = \frac{1}{k}(X_a \cos\theta - Y_a \sin\theta) \\ Y_n = \frac{1}{k}(Y_a \cos\theta + X_a \sin\theta) \\ Z_n = Z_0 - \theta \end{cases} \tag{7}$$

对于一组特殊的初始值：

$$\begin{cases} X_a = k \approx 0.607253 \\ Y_a = 0 \\ Z_0 = \theta \end{cases}$$

得到的结果为：

$$\begin{cases} X_n = \cos\theta \\ Y_n = \sin\theta \\ Z_n = 0 \end{cases}$$

它的工作过程是使Z_i不断地趋近于 0，这种工作模式就是旋转工作模式。

1.2 CORDIC 向量模式

另外一种工作模式是向量工作模式，向量模式就是不断地使Y_i趋近于 0，如图 3 所示。

在向量模式下，CORDIC 迭代n次（$n \to \infty$）得到的最终结果为：

$$\begin{cases} X_n = \dfrac{1}{k}\sqrt{X_a{}^2 + Y_a{}^2} \\ Y_n = 0 \\ Z_n = \theta \end{cases} \tag{8}$$

这时，式(6)中，当$Y_i \geq 0$时，$S_i = -1$；当$Y_i < 0$时，$S_i = +1$。

对于一组给定的初始化值：

$$\begin{cases} X_a = X \\ Y_a = Y \\ Z_a = 0 \end{cases}$$

得到的迭代结果为：

$$\begin{cases} X_n = \dfrac{1}{k}\sqrt{X^2 + Y^2} \\ Y_n = 0 \\ Z_n = \arctan(\dfrac{Y}{X}) \end{cases}$$

2 CORDIC 算法的 FPGA 实现

2.1 总体方案设计

按 J.S.Walther 提出的迭代序列：0，1，2，……，N-1，对于不同的 N 值，旋转后可以获得最大的角度如表 1 所示

由表 1 可以看出，$-99.88^{\circ} \le \lim_{N\to\infty}\sum_{i=0}^{N}\theta_i \le 99.88^{\circ}$，也就是说覆盖的角度只有-99.88°～+99.88°，无法覆盖完整的周期[-180°，+180°)。为了确保 CORDIC 算法收敛，其所有旋转角度之和必须大于实际需要旋转的角度，所以我们必须将输入的角度进行预处理，利用三角函数的对称性把输入角度限定在一定范围内（旋转模式：-90°～+90°；向量模式：0~+90°）。由于存在预处理单元，所以通过 CORDIC 迭代得到的输出并不是最终的结果，还需要通过一个后处理单元进行转化进而得到预期的输出。

表 1 不同 N 值对应的角度范围

N	0	1	2	3	4	5	6
$\max(\sum_{i=0}^{N}\theta_i)$	45°	71.56°	85.60°	92.73°	96.30°	98.09°	98.99°
N	7	8	9	10	11	12	≥13
$\max(\sum_{i=0}^{N}\theta_i)$	99.44°	99.67°	99.77°	99.83°	99.85°	99.87°	99.88°

通过以上分析，我们把设计分为 3 个子模块，分别是：CORDIC_PRE 模块、CORDIC 模块和 CORDIC_POST 模块，这三个子模块以及顶层 CORDIC_TOP 模块之间的关系如图 4 所示。

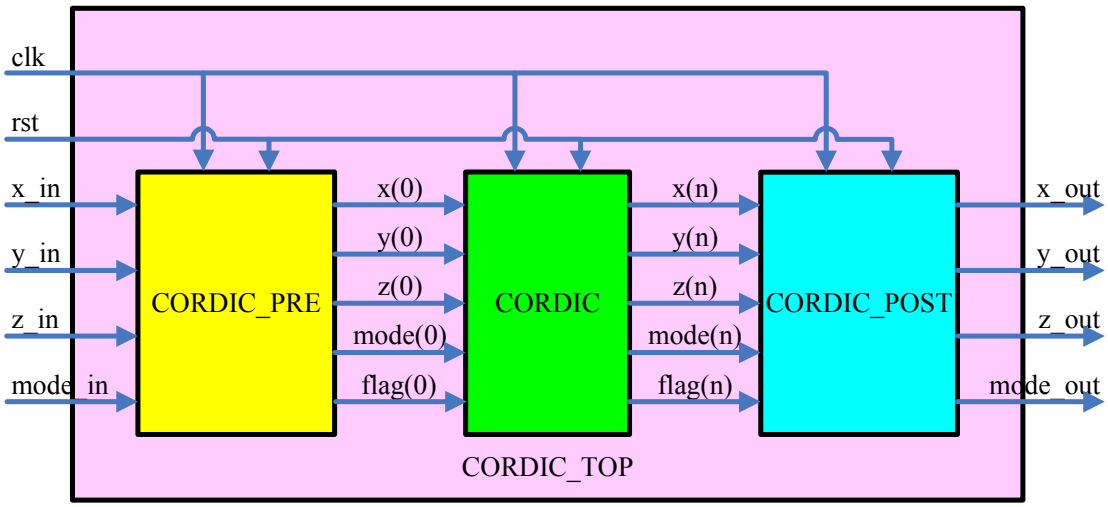

图 4 总体方案设计

增加迭代次数可以提高运算精度，但这无疑增加了运算的复杂度，影响到信号处理数据流的速度，因此，必须在运算精度和信号处理速度之间进行权衡。这里我们为了提高精度，对所有内部信号及输出信号都用 20bits 的补码表示。

输入输出信号说明如表 2 所示

表 2 CORDIC 模块的接口表

接口名称	宽度	类型	说明
clk	1	input	系统时钟
rst	1	input	复位信号
mode_in	1	input	输入模式。旋转模式：mode_in=0；向量模式：mode_in=1
x_in	16	input	输入
y_in	16	input	输入
z_in	16	input	输入
mode_out	1	output	输出模式。旋转模式：mode_out=0；向量模式：mode_out=1
x_out	20	output	输出
y_out	20	output	输出
z_out	20	output	输出

2.2 预处理及后处理

2.2.1 旋转模式

工作在旋转模式时，输入 x_in，y_in 是常数，输入 z_in 为待求正余弦值的角度，覆盖范围为[-180°，+180°)（I，II，III，IV 象限）；而旋转覆盖的角度只有-99.88°～+99.88°（I，IV 象限），所以我们需要在迭代开始前把 z_in 所在象限转化到 I，IV 象限。

迭代后的结果需要根据 z_in 的正负以及三角函数的性质进行后处理。

具体的预处理与后处理方法如表 3 所示。

表 3 旋转模式的预处理和后处理

预处理	后处理

if (z_in≥0) then
z(0)=z_in-90°;
else
z(0)=z_in+90°;
endif
	if (z_in≥0) then
	x_out=cos(z_in)=-sin(z_in-90°)=-sin(z(0))=-y(n);
	y_out=sin(z_in)=cos(z_in-90°)=cos(z(0))=x(n);
	else
	x_out=cos(z_in)=sin(z_in+90°)=sin(z(0))=y(n);
	y_out=sin(z_in)=-cos(z_in+90°)=-cos(z(0))=-x(n);
	endif

2.2.2 向量模式

工作在向量模式时，输入 z_in 是常数，输入点(x_in,y_in)为直角坐标系下的某点坐标，覆盖范围为 I，II，III，IV 四个象限；而旋转覆盖的角度只有-99.88°～+99.88°（I，IV 象限），所以我们需要在迭代开始前判断点(x_in,y_in)所在的象限，并将它转化到 I 象限；另外，由于我们是判断 y(i)的正负来确定 S(i)的正负，角度小于 45°的角，角度的变化对于 y(i)值的影响比较明显，所以如果所求向量角度在 45°～90°的范围内，需要先将 x_in 和 y_in 互换，转换到 0°～45°的范围内进行计算，这样收敛的速度比较快，在相同迭代次数的条件下可以减少误差。

向量模式下的后处理主要有两个目的。其一是把 x(n) 乘以 k 得到 $((x\_in)^2+(y\_in)^2)^{1/2}=x\_out$；其二是由输入点(x_in,y_in)的象限信息以及是否进行交换得到最终的角度信息z_out。

这里我们通过一个技巧来实现目的一中的乘法，假定temp=x(n)/2+x(n)/8-x(n)/64-x(n)/512，则 x_out=temp-temp/4096=x(n)*0.60727，这样我们之需要进行加/减法及简单的移位操作就实现了原来的乘法运算，其误差为 0.0034%。

具体的预处理与后处理方法如表 4 所示。

表 4 向量模式的预处理和后处理

预处理	后处理

if (x_in≥0) then
x_abs_temp=x_in;
x_flag=1'b0;
else
x_abs_temp=-x_in;
x_flag=1'b1;
endif
if (y_in≥0) then
y_abs_temp=y_in;
y_flag=1'b0;
else
y_abs_temp=-y_in;
y_flag=1'b1;
endif
if (y_abs_temp>x_abs_temp) then
swap=1'b1;
x_abs=y_abs_temp;
y_abs=x_abs_temp;
else
swap=1'b0;
x_abs=x_abs_temp;
y_abs=y_abs_temp;
endif
	①
	temp=x(n)/2+x(n)/8-x(n)/64-x(n)/512;
	x_out=temp-temp/4096=x(n)*0.60727;
	②
	case(y_neg,x_neg,swap)
	000:
	z_out=z(n);
	001:
	z_out=90°-z(n);
	010:
	z_out=180°-z(n);
	011:
	z_out=90°+z(n);
	100:
	z_out=-z(n);
	101:
	z_out=-90°+z(n);
	110:
	z_out=-180°+z(n);
	111:
	z_out=-90°-z(n);
	endcase

2.3 CORDIC 迭代

表 5 旋转模式和向量模式的迭代过程

旋转模式	向量模式

for i=0 to num_iteration
if (z(i)>=0) then
x(i+1)=x(i)-y(i)/2^i;
y(i+1)=y(i)+x(i)/2^i;
z(i+1)=z(i)-arctan(1/2^i);
else
x(i+1)=x(i)+y(i)/2^i;
y(i+1)=y(i)-x(i)/2^i;
	for i=0 to num_iteration
	if (y(i)>=0) then
	x(i+1)=x(i)+y(i)/2^i;
	y(i+1)=y(i)-x(i)/2^i;
	z(i+1)=z(i)+arctan(1/2^i);
	else
	x(i+1)=x(i)-y(i)/2^i;
	y(i+1)=y(i)+x(i)/2^i;

z(i+1)=z(i)+arctan(1/2^i);
end if
end for
	z(i+1)=z(i)-arctan(1/2^i);
	end if
	end for

通过第一节的分析，我们可以把两种模式下的迭代过程用表 5 的伪代码来表示。

为了提高运算速度，可以采用流水线结构。流水线处理是高速设计中的一个常用设计手段。如果某个设计的处理流程分为若干步骤，而且整个数据处理是“单流向”的，既没有反馈或者迭代运算，前一个步骤的输出是下一个步骤的输入，则可以考虑采用流水线设计方法提高系统的工作频率。流水线设计的结构示意图如图 5 所示。

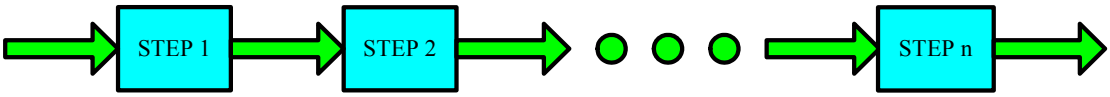

图 5 流水线设计的结构示意图

流水线基本结构是将适当划分的 *n* 个操作步骤单流向串联起来。流水线操作的最大特点和要求是，数据流在各个步骤的处理，从时间上看是连续的，如果将每个操作步骤简化假设为通过一个 D 触发器（就是用寄存器打一个节拍），那么流水线操作就类似一个移位寄存器组，数据流依次流经 D 触发器，完成每个步骤的操作。流水线设计时序示意图如图 6 所示。

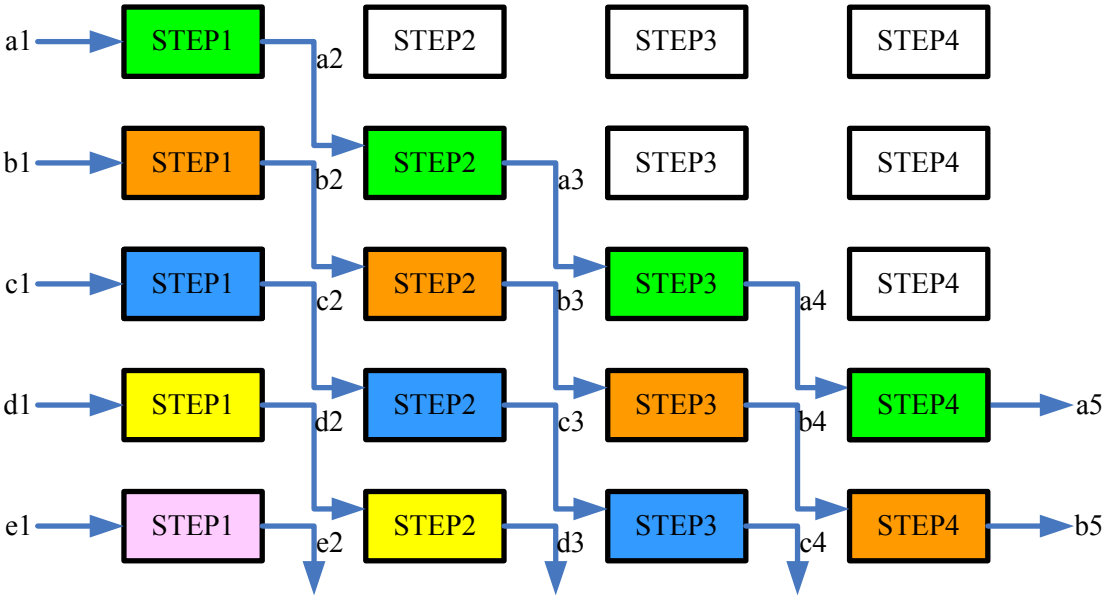

图 6 流水线设计时序示意图

流水线结构充分利用了硬件内部并行性，增加数据处理能力。这种流水线作业是在几个步骤中执行运算的功能单元的序列。每个功能单元接受输入，生成的输出则是缓冲器存储的输出。在流水线作业中，一级的输出变成下一级的输入，从而使所有各级的输入输出都是并行运行的。这种配置可以很大程度上提高运算速度。

实现流水线结构的方法很简单，只要在每个运算部件（包括乘法器和加减法器）的输出以及系统的输入输出之间加上寄存器缓存即可，利用流水线技术

的 CORDIC 迭代实现框图如图 7 所示，该图所描述的是 CORDIC 的第 i 次迭代。

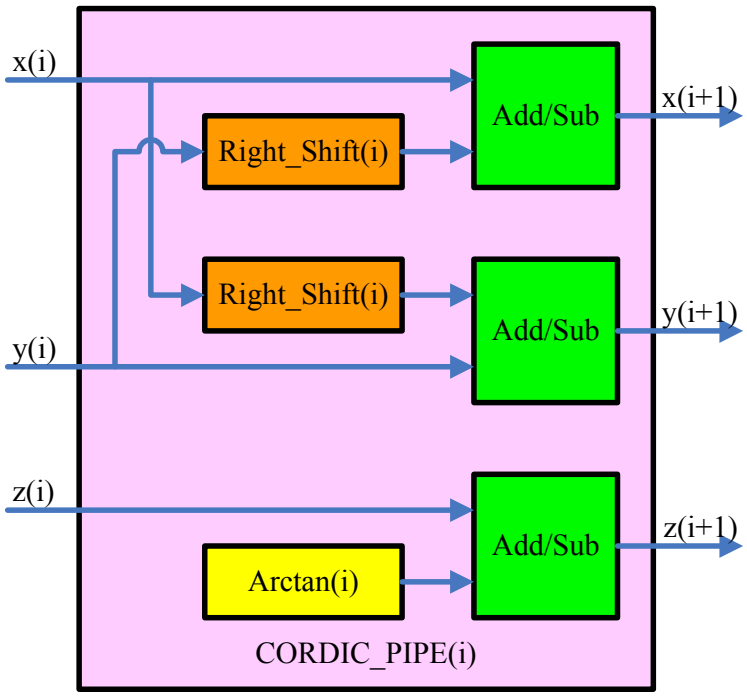

图 7 CORDIC 第 i 次迭代硬件结构框图

对于 Right_Shift(i) 模块，我们可以利用连接操作符，通过 y(i)/2^i={{i{y(i)[19]}},y(i)[19:i]}，x(i)/2^i={{i{x(i)[19]}},x(i)[19:i]}这两条语句直接实现。

表 6 Arctan(i)对应的值

i	角度（°）	Arctan(i)
0	45	20'h20000
1	26.565051	20'h12e40
2	14.036243	20'h09fb4
3	7.125016	20'h05111
4	3.576334	20'h028b1
5	1.789910	20'h0145d
6	0.895173	20'h00a2f
7	0.447614	20'h00518
8	0.223810	20'h0028c
9	0.111905	20'h00146
10	5.59528E-02	20'h000a3
11	2.79764E-02	20'h00051
12	1.39882E-02	20'h00029
13	6.99411E-03	20'h00014
14	3.49705E-03	20'h0000a
15	1.74852E-03	20'h00005
16	8.74264E-04	20'h00003
17	4.37132E-04	20'h00001

18	2.18566E-04	20’h00000
19	1.09283E-04	20’h00000
20	5.46415E-05	20’h00000

对于 Arctan(i)模块，我们可以用一个 case 语句来实现。表 6 是针对不同的迭代次数 i 时，Arctan(i)所对应的值。从表 6 中我们可以发现，当采用 20bits 表示 Arctan(i)时，CORDIC 只需要进行 18 次迭代（i 从 0 到 17）。

3 仿真结果与分析

3.1 旋转模式

工作在该模式下时，设输入的角度依次为-180°，-150°，-120°，-90°，-60°，-30°，0°，30°，60°，90°，120°，150°，得到的实际正余弦值如图 8 所示。

下面以-30°为例分析一下输出结果。因为已假定输入角度采用 16 位补码表示，覆盖范围为[-180°，+180°)。+30°表示为 30*(2^16/360)≈5461(dec)=1555(hex)，所以-30°表示为 1555(hex)的补码，即 eaab(hex)。由于为了提高精度，输出的 cos、sin 采用 20 位补码表示，分别为 6eda1(hex)和 c0019(hex)（如图 8 所示）。6eda1(hex)对应 0.866029，c0019(hex)对应 -0.499952，与其对应的理论值 cos(-30°)=0.866025，sin(-30°)=-0.5 相比较误差已相当小，能够满足一般的应用需要。其它实际值与理论值之间的差异比较详见表 7。

表 7 旋转模式下理论值与实际值

旋转模式 mode_in=0	**角度**	**理论值** cos **理论值** sin	z_in	x_out y_out	**实际值** cos **实际值** sin
	-180°	-1.000000 0.000000	16’h8000	20’h8000b 20’h00005	-0.999979 0.000009
	-150°	-0.866025 -0.500000	16’h9555	20’h9125f 20’hc0018	-0.866029 -0.499954
	-120°	-0.500000 -0.866025	16’haaab	20’hc0019 20’h91260	-0.499952 -0.866027
	-90°	0.000000 -1.000000	16’hc000	20’hffffe 20’h8000a	-0.000003 -0.999980
	-60°	0.500000 -0.866025	16’hd555	20’h3ffe7 20’h9125f	0.499952 -0.866029
	-30°	0.866025 -0.500000	16’heaab	20’h6eda1 20’hc0019	0.866029 -0.499952
	0°	1.000000 0.000000	16’h0000	20’h7fff5 20’hfffffb	0.999979 -0.000009
	30°	0.866025 0.500000	16’h1555	20’h6eda1 20’h3ffe8	0.866029 0.499954
	60°	0.500000 0.866025	16’h2aab	20’h3ffe7 20’h6eda0	0.499952 0.866027

	90°	0.000000 1.000000	16’h4000	20’h00002 20’h7fff6	0.000003 0.999980
	120°	-0.500000 0.866025	16’h5555	20’hc0019 20’h6eda1	-0.499952 0.866029
	150°	-0.866025 0.500000	16’h6aab	20’h9125f 20’h3ffe7	-0.866029 0.499952

图 8 旋转模式下的仿真波形

3.2 向量模式

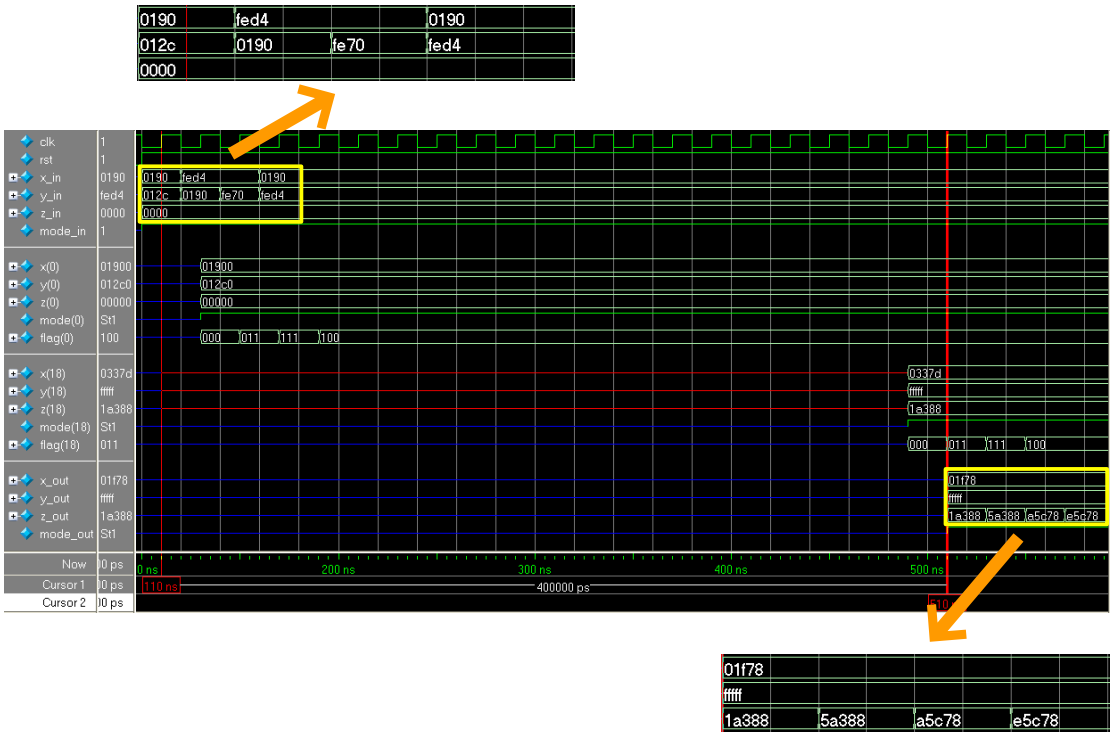

图 9 向量模式下的仿真波形

工作在该模式下时，设输入的坐标值依次为(400，300)，(-300，400)，(-300，-400)和(400，-300)，得到的实际结果如图 9 所示。

下面以输入坐标(-300，-400)为例分析一下输出结果。输出还是采用 20 位补码表示，分别为 01f78(hex)和 a5c78(hex)（如图 9 所示）。01f78(hex)对应 503.5，c0019(hex)对应-126.872863°，而理论值分别是 500 和-126.869897°，从中我们可以发现角度的误差较小而幅值的误差较大。如果要提高幅值的精度，可以通过增加位宽和迭代次数来实现。

4 结束语

本文探讨了双模式 CORDIC 算法的硬件实现，它具有速度快、精度高、实现简单等优点，在合理地选择迭代次数和位宽的条件下，能够满足速度和精度的要求，因此具有十分重要的工程研究和应用意义。本设计还可以做些适当的改进（如增加位宽，增加迭代次数），以满足更高的要求。

参考文献

[1] Andraka R. A Survey of Cordic Algorithms for FPGA Based Computers[M]. 2003.

[2] VOLDER J E. The CORDIC trigonometric computing technique[J]. IRE Trans. Electronic Computers, 1959,EC-8(3): 330-334.

[3] 李滔, 韩秋月. 基于流水线CORDIC算法的三角函数发生器[J]. 电子技术应用, 1999,(6): 45~49.

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

[5] ANDRAKA R. A survey of CORDIC algorithms for FPGA based computers[C]. Sixth International Symposium on Field Programmable Gate Arrays. Monterey, 1998.

[6] Wassatsch A, et al. Area Minimization of Redundant CORDIC Pipeline Architectures[C]. IEEE International Conference on Computer Design: VLSI in Computers and Processors. 1998,: 136-141.

原创性声明

本人郑重声明：此处提交的论文《双模式 CORDIC 算法的 FPGA 实现》，是本人在导师指导下，在哈尔滨工业大学期间进行研究工作所取得的成果。据本人所知，论文中除已注明部分外不包含他人已发表或撰写过的研究成果。对本文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。本声明的法律效果将完全由本人承担。

作者签字　王睐来　　日期：2008 年 4 月 25 日