

基于CORDIC算法的32位浮点三角超越函数之正余弦函数的FPGA实现

The implementation of 32 bits floating trigonometric transcendental

—sin&cos function on FPGA based on the CORDIC algorithm

李全 陈石平 付佃华

桂林电子科技大学 通信与信息工程系 541004

摘要: 本文在传统CORDIC算法的基础之上,通过增加迭代次数,对参数进行了优化筛选,提高了运算精度,使设计出的软核能够在精度要求较高的场合中运行,如实时语音、图像信号处理、滤波技术等。输出数据经过IEEE-754标准化处理,能够直接兼容大多数处理器,扩展了其应用范围。最终在Altera公司Nios II处理器中通过增加自定义指令的方式完成了硬件实现。

关键字: CORDIC ,自定义指令, IEEE-754标准化处理。

Abstract: This paper improve the calculation precision through increasing the iteration times, optimizing and selecting the arguments base on the traditional CORDIC algorithm, making the designed soft-core adapted to higher precision required occasion such as real-time voice, image and signal processing, filter technique and so on. The output data can be directly compatible with most of the processors by the IEEE-754 standard processing, so its application area is widely extended. Finally we accomplish the hardware implementation by adding custom instruction on the Nios II processor of Altera corporation.

Keywords: CORDIC, custom instruction, IEEE-754 standard processing .

引言

浮点超越函数的应用领域十分广泛,涉及航空航天、机器人技术、实时语音、图像信号处理、滤波技术、FFT变换等领域。因此,设计并实现浮点三角超越函数是非常重要的。硬件实现的超越函数算法,按照数学公式和对应的实现方式的不同,可以分为查表法、多项式近似法、基于查表的多项式结合方法、有理数近似和逐位法五类。经过对这些算法进行分析和比较,本文选择CORDIC作为超越函数的算法,并用Altera公司的Cyclone II芯片完成硬件实现。

1 CORDIC原理

CORDIC (Coordinate Rotation Digital Computer) 算法即坐标旋转数字计算方法,是J. D. Volder^[1]于1959年首次提出,主要用于三角函数、双曲线、指数、对数的计算。该算法通过基本的加和移位运算代替乘法运算,使得矢量的旋转和定向的计算不再需要三角函数、乘法、开方、反三角、指数等函数。

1.1 圆周系统

CORDIC算法包含圆周系统,线性系统,双曲系统三种旋转系统。本文仅以圆周系统推导如下:

该系统完成的是一个平面坐标旋转,如图1所示。从图1中可以看出,将向量 (X_i, Y_i) 旋转 θ 角,得到一个新的向量 (X_j, Y_j) ,那么有:

$$\begin{aligned} X_j &= R \cos(\theta + \beta) = X_i \cos \theta - Y_i \sin \theta \\ Y_j &= R \sin(\theta + \beta) = X_i \sin \theta + Y_i \cos \theta \end{aligned} \quad (1)$$

其中R为圆周的半径， θ 为旋转角度。把上式化为矩阵形式，平面旋转定义如下：

$$\begin{bmatrix} X_j \\ Y_j \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \end{bmatrix} \quad (2)$$

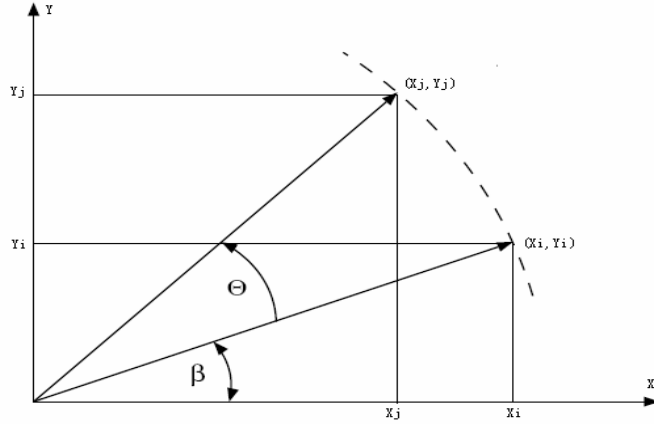


图1 坐标旋转图

使用迭代的方法，旋转的角度可以在多步之内完成，每一步的旋转完成其中的一小部分，多步之后将会完成一个平面旋转。旋转等式由式(2)定义，消除 $\cos \theta_n$ 因子得到单步旋转等式为：

$$\begin{bmatrix} X_{n+1} \\ Y_{n+1} \end{bmatrix} = \begin{bmatrix} \cos \theta_n & -\sin \theta_n \\ \sin \theta_n & \cos \theta_n \end{bmatrix} \begin{bmatrix} X_n \\ Y_n \end{bmatrix} = \cos \theta_n \begin{bmatrix} 1 & -\tan \theta_n \\ \tan \theta_n & 1 \end{bmatrix} \begin{bmatrix} X_n \\ Y_n \end{bmatrix} \quad (3)$$

式(3)中每一步的旋转角度为： $\theta_n = \arctan\left(\frac{1}{2^n}\right)$ ，且 $\sum_{n=0}^{\infty} S_n \theta_n = \theta$ ， $S_n = \{-1; +1\}$ ，即所有迭代角之和必须等于旋转角度。对于每一步旋转角度 Z_n 如下：

$$Z_n = \theta - \sum_{i=0}^{n-1} S_i \theta_i \quad (4)$$

其中 S_n 为 Z_n 的符号函数， $S_n = \begin{cases} -1 & \text{if } Z_n < 0 \\ +1 & \text{if } Z_n \geq 0 \end{cases}$ (5)

综合式(3)、式(4)和式(5)得到：

$$\begin{bmatrix} X_{n+1} \\ Y_{n+1} \end{bmatrix} = \cos \theta_n \begin{bmatrix} 1 & -S_n 2^{-n} \\ S_n 2^{-n} & 1 \end{bmatrix} \begin{bmatrix} X_n \\ Y_n \end{bmatrix} \quad (6)$$

再通过 N 次迭代后：

$$\begin{bmatrix} X_j \\ Y_j \end{bmatrix} = \prod_{n=0}^N \cos \theta_n \begin{bmatrix} 1 & -S_n 2^{-n} \\ S_n 2^{-n} & 1 \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \end{bmatrix} = K^* \prod_{n=0}^N \begin{bmatrix} 1 & -S_n 2^{-n} \\ S_n 2^{-n} & 1 \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \end{bmatrix} \quad (7)$$

将比例因子从迭代公式中提取出来，定义 K 为增益因子，则有：

$$K = \frac{1}{P} = \prod_{n=0}^N \cos\left(\arctan\left(\frac{1}{2^n}\right)\right) = \prod_{n=0}^N \frac{1}{\sqrt{1+2^{-2n}}} \approx 0.607253 \quad (8)$$

具体增益 K 取决于迭代次数 N 。对于所有的初始向量和所有的旋转角度而言， K 是一个常数。通常把 K 称作聚焦常数^[2]，式 (8) 中常数 P 为 K 的倒数。

1.2 计算模式

CORDIC 算法有旋转模式和向量模式^[3] 两种计算模式。

1.2.1 旋转模式

在旋转模式中， Z 为初始化需要旋转的角，当 Z 旋转变为 0 时，公式变化如下：

$$\begin{aligned} X_{n+1} &= X_n - S_n Y_n 2^{-n} \\ Y_{n+1} &= Y_n + S_n X_n 2^{-n} \\ Z_{n+1} &= Z_n - S_n \tan^{-1}(2^{-n}) \end{aligned} \quad (9)$$

经过 N 次迭代后，CORDIC 公式的输出变为：

$$\begin{aligned} X_{n+1} &= P[X_0 \cos(Z_0) - Y_0 \sin(Z_0)] \\ Y_{n+1} &= P[Y_0 \cos(Z_0) + X_0 \sin(Z_0)] \quad \text{其中 } P = \prod_{n=0}^N \sqrt{1+2^{-2n}} \\ Z_{n+1} &= 0 \end{aligned} \quad (10)$$

如果 $X_0 = 1/P, Y_0 = 0, Z_0 = \alpha$ ，那么 N 次迭代后 CORDIC 公式的输出变为：

$$[X_{n+1}, Y_{n+1}, Z_{n+1}] = [\cos(\alpha), \sin(\alpha), 0] \quad (11)$$

从式(11)的分析可以看出，CORDIC 算法在圆周系统的旋转模式可以用来计算一个输入角的正弦、余弦和正切，此外还可以将极坐标变换为平面坐标。

1.2.2 向量模式

向量模式将输入向量通过一个特定的角 Y 变为 0。这种模式下的 CORDIC 算法跟旋转模式差不多，区别是旋转的方向取决于 Y 而不是 Z 的符号。

那么 N 次迭代后 CORDIC 公式的输出变为：

$$\begin{aligned} X_{n+1} &= P\sqrt{X_0^2 + Y_0^2} \\ Y_{n+1} &= 0 \\ Z_{n+1} &= Z_0 + \tan^{-1}(Y_0/X_0) \end{aligned} \quad \text{其中 } P = \prod_{n=0}^N \sqrt{1+2^{-2n}} \quad (12)$$

如果 $Z_0 = 0$ ，对于给定的 X_0 和 Y_0 ， N 次迭代后 CORDIC 公式的输出变为：

$$[X_{n+1}, Y_{n+1}, Z_{n+1}] = [P\sqrt{X_0^2 + Y_0^2}, 0, \tan^{-1}(Y_0/X_0)] \quad (13)$$

从式(13)可以看出，CORDIC 算法在向量模式可以计算给定向量 (X, Y) 的长度和角度，即从平面坐标到极坐标的变换。

2 IEEE-754标准浮点结构^[4]

IEEE-754标准有单精度、双精度和扩展精度三种浮点数表示格式。单精度浮点数由符号位，指数偏置位 $e = E + \text{bias}$ (E 是未偏置的指数) 和小数 $f = .b_{22}b_{21}b_{20} \dots b_0$ 三个部

分组成。单精度浮点数格式如图2所示。



图 2 浮点单精度数表示格式

图中的 Sign 是符号位, Exp 是指数位, Fraction 是尾数位, 单精度中的指数位与尾数位之间有一个隐含整数位。本文使用 32 位单精度浮点小数表示运算。

未偏置的指数E应该包含最大数 E_{max} 和最小数 E_{min} 之间的所有数, 其中保留 $E_{min}-1$ 的 ± 0 编码和非标准化数, 以及 $E_{max}+1$ 的 $\pm\infty$ (正负无穷大) 编码和NaNs (不是一个数)。上述的各参数如表1所示。

表 1 参数格式表

| Parameter | Format |
|------------------------|--------|
| P | 24 |
| E_{max} | +127 |
| E_{min} | -126 |
| Exponent bias | +127 |
| Exponent width in bits | 8 |
| Format width in bits | 32 |

图 2 所示的是 32 位单精度浮点数格式, 数 X 组成域可以推断出 X 的二进制值 v 表示如下:

- (1) NaN(不是一个数): 若 $e=255$, 且 $f \neq 0$, 则 $v = \text{NaN}$;
- (2) $\pm\infty$ 表示正负无穷大: 若 $e=255$, 且 $f=0$, 则 $v = (-1)^s * \infty$;
- (3) 规格化数: $0 < e < 255$, 值 $v = (-1)^s 2^{e-127} (1.f)$;
- (4) DNRM(非规格化数): 如果 $e=0$, 但 $f \neq 0$, 则 $v = \text{DNRM}$;
- (5) ± 0 : 如果 $e=0$ 且 $f=0$, 则 $v = (-1)^s * 0$.

3 CORDIC 内核及前后处理单元设计

3.1 CORDIC 内部结构

在迭代式架构下, 内部 CORDIC 结构^[1]如图 3 所示。

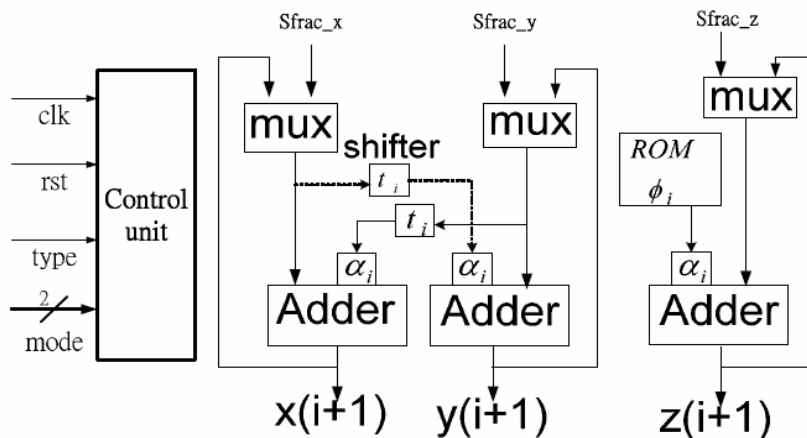


图 3 经典 CORDIC 硬件实现架构

图 3 中架构由控制单元，选择器，移位寄存器，ROM，加法器组成。系统的控制单元采用有限状态机的实现形式，进行迭代和数据格式转换的流程控制。ROM 中存放的是预先计算好的反正切数据值，供迭代运算中调用。从架构中可以看出只有移位与加减运算，因此特别适合于用硬件实现。每次迭代运算后，再将数据返回，直到足够的次数执行完成后，才将结果输出。每次迭代之前要对角度值 Z 进行判断，根据 Z 的符号来决定图 3 中 a_i 的符号。

实现该部分的关键代码如下：

```

if(~z[31])
begin
x <= x - (y >>> counter);
y <= y + (x >>> counter);
z <= z - atan;
end
else
begin
x <= x + (y >>> counter);
y <= y - (x >>> counter);
z <= z + atan;
end

```

3.2 前置处理单元

由于迭代本身的限制，最大旋转角度用公式表示为：

$$|\theta_{\max}| = \sum_{i=0}^n \arctan\left(\frac{1}{2^i}\right) \approx 99.9^\circ \quad (14)$$

为了扩展输入角的范围，需要对输入角进行预处理，使其落在第一象限范围之内。即对输入角度进行 90° 取模，根据所处的象限进行符号判断和正余弦交换。在 CORDIC 内核设计中，对 $0 \sim 360^\circ$ 范围内的角度用一个 32 位的二进制整数来表示，即 $1^\circ \approx 101101100000101101100000$ （即十进制数 11930464）。

3.3 后置处理单元^[5]

后置处理将内部 CORDIC 计算出来的结果在输出前转换成 IEEE-754 标准格式，即 $((-1)^s 2^{e-127} (1.f))$ 形式，其结构如图 4 所示。

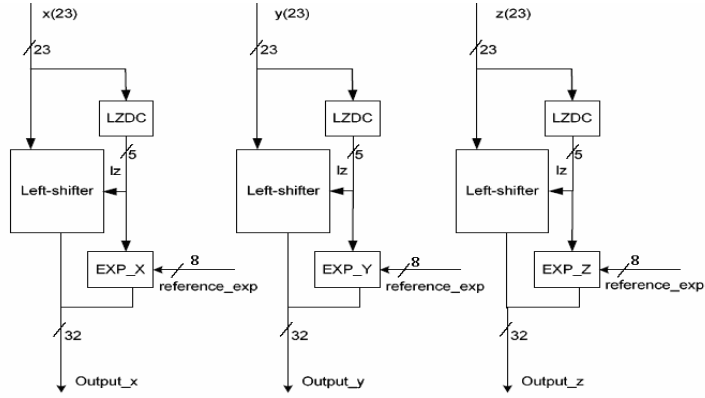


图 4 后置处理单元硬件结构

图 4 中的 LZDC (Leading zero detection circuit) 为前导零检测电路, 其功能是用来统计数据从最高位开始连续“0”的个数, 再根据它把计算出来的数值转换成 IEEE-754 标准格式。在转换过程中需要对尾数进行左移, 才能表示为 $(-1)^s 2^{e-127} (1.f)$, 此位移量再与前置处理器单元传来的参考指数值相减, 得到输出指数值。符号位直接由前置处理过程得到。最后将符号位、指数数值、小数按位连接得到 IEEE-754 格式的数据, 并从硬件接口输出端输出。

4 设计及软件仿真波形

用 Quartus II 对本设计进行了软件波形仿真, 选取角度值分别为 0° 、 30° 、 88.8° 和 359° (即 -1°), 仿真波形如图 5~图 8 所示。

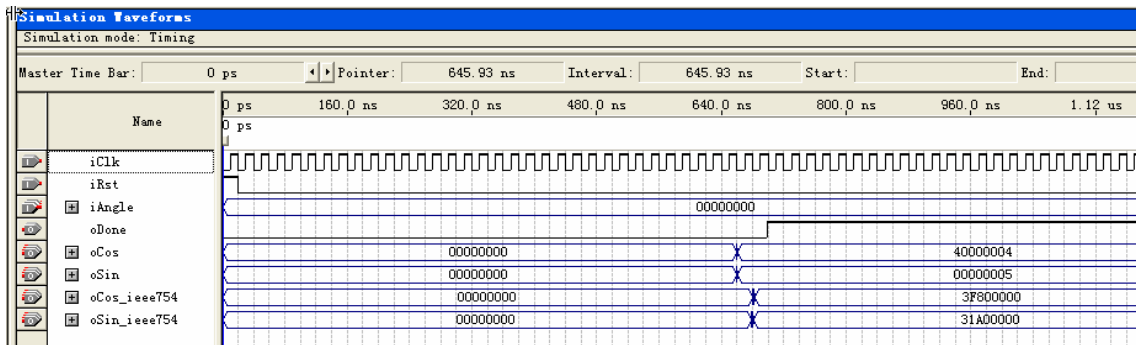


图 5

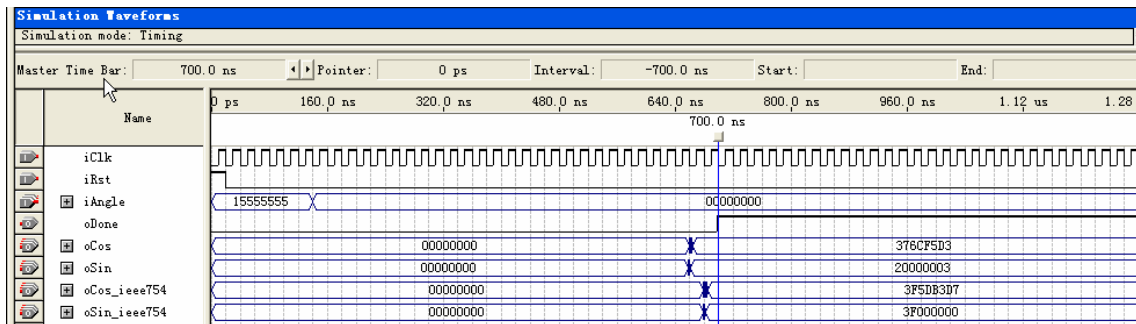


图 6

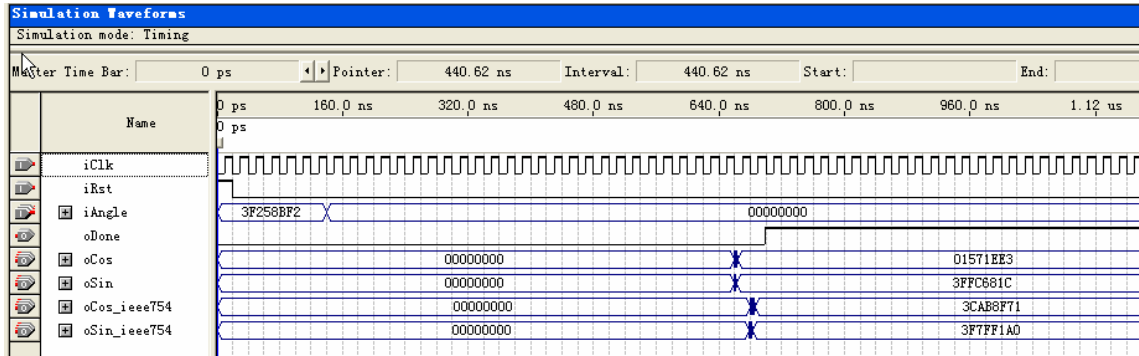


图 7

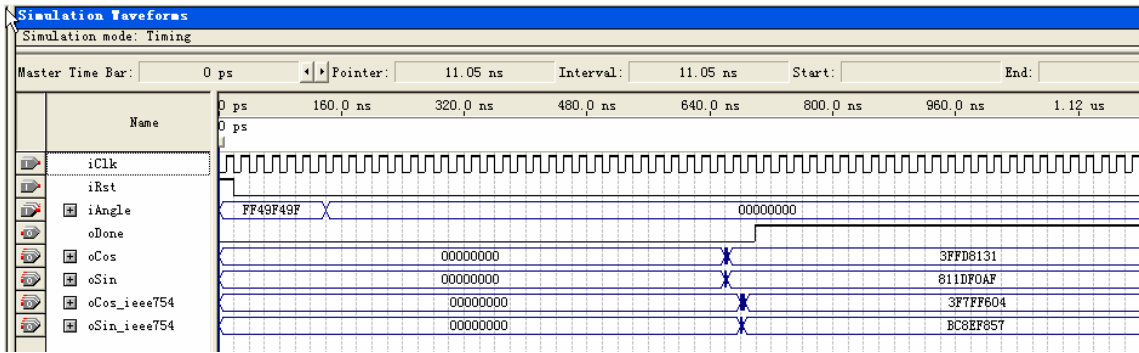


图 8

以 88.8° 为例, $iAngle$ 的十进制数为 $(2^{32}/360) \times 88.8 = 1059425266$ (小数部分忽略), 十六进制表示为 $3F258BF2$ 。 $oCos$ 的内部仿真结果十六进制值为 $3FFD8131$, 对应的 IEEE-754 标准格式值 $oCos_IEEE754$ 为 $3F7FF804$, 即十进制数 0.020942 , 正弦的计算过程同理。

仿真结果对照如表 2 所示。

表 2 仿真结果对照表

| 输入 (度) | 内部表示 | Cos(IEEE 表示) | Sin(IEEE 表示) | Cos | Sin |
|--------------|------------|--------------|--------------|----------|-----------|
| 0 (图 5) | 0x00000000 | 0x3f800000 | 0x31a00000 | 1.000000 | 0.000000 |
| 30 (图 6) | 0x15555555 | 0x3f5db3d7 | 0x3f000000 | 0.866025 | 0.500000 |
| 88.8 (图 7) | 0x3f258bf2 | 0x3cab8f71 | 0x3f7ff1a0 | 0.020942 | 0.999780 |
| 359(-1)(图 8) | 0xff49f49f | 0x3f7ff604 | 0xbc8ef857 | 0.999847 | -0.017452 |

从表中可以看出, 在 $(0 \sim 360^\circ)$ 输入范围内, 用本设计计算仿真出的结果误差小于 10^{-6} , 在单精度表示范围内误差为 0, 且输出表示为标准 IEEE754 类型。一次计算所需时钟周期数为 34。

5 自定义指令

Altera 公司 Nios II 处理器增加了多达 256 条用户自定义指令, 这是其它 SOC 系统无法比拟的。用户自定义就是让 Nios II 软核完成一个用 HDL 描述的电路模块功能。用户自定义指令可以在几个时钟周期之内完成复杂算法的处理能力, 访问存储器或系统外的逻辑单元, 加快专项任务的执行, 以达到优化目的。通过用户自定义指令可以把系统中用软件处理中的耗时多的关键算法用硬逻辑电路来实现, 大大提高处理器的效率。由于

CORDIC内核的特性，正余弦函数可以同时计算出来，因此适合于采用扩展用户自定义指令来实现，即用一个硬件模块来完成两种函数的计算。通过一个附加的参数n来决定输出是正弦或余弦，这样可以节约一半的硬件资源。本文在Nios II中增加了正余弦自定义指令，其宏定义代码如下：

```
#define MYCORDIC_N 0x00000002
#define MYCORDIC_N_MASK ((1<<1)-1)

#define FLOAT_COS(A)  __builtin_custom_fni(MYCORDIC_N+(1&MYCORDIC_N_MASK), (A))
#define FLOAT_SIN(A)  __builtin_custom_fni(MYCORDIC_N+(0&MYCORDIC_N_MASK), (A))
```

加入指令后，在Nios II的IDE中进行软硬件同时调试，相应的C语言测试代码见附录1，给出对比结果如图9和图10所示。

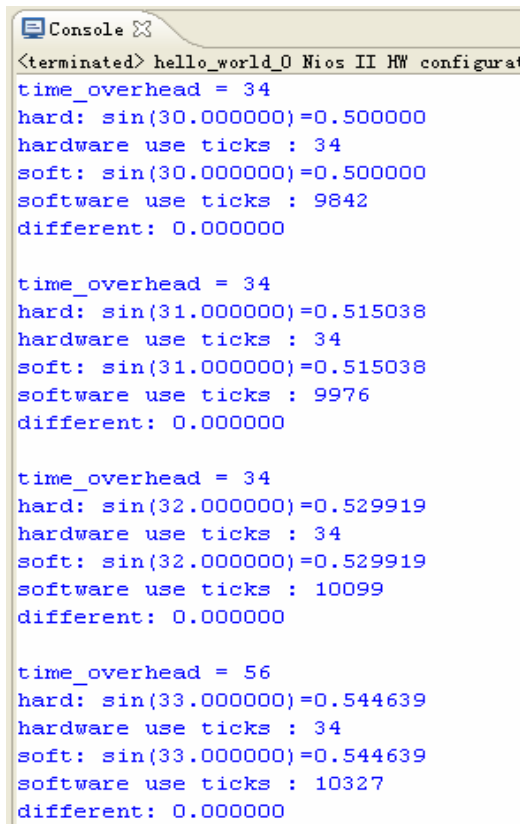


图 9

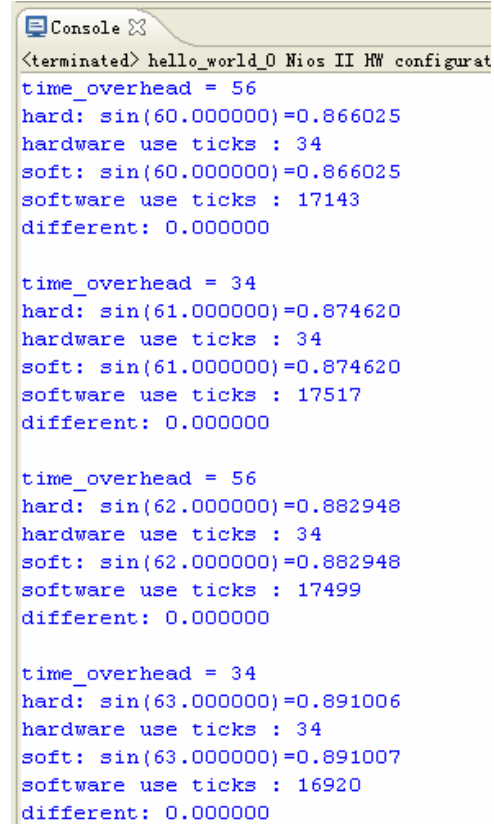


图 10

本系统硬件验证平台采用 DE2 开发板，其中 Nios II 处理器配置为 full 功能，分别使用 16K 数据 cache 和 16K 指令 cache，增加了硬件乘法器和硬件除法器，开启了流水线功能，已达到了 Nios II 软核处理器的最高性能。

从结果对比中看出，使用硬件指令后对完全相同的正余弦角度进行计算时，其运算速度大大提高。需要说明的是，软件运算随着输入角度的不同所使用的计算时间有很大差别，而使用硬件指令运算是，其计算时间基本不变。详细对比如表 3 所示。

表3 软硬件运算所耗时钟周期数对比表

| 计算角度 | 软件时钟周期数 | 硬件时钟周期数 | 软硬时钟周期数之比 |
|------|---------|---------|-----------|
| 30 | 9842 | 34 | 289 |
| 31 | 9976 | 34 | 293 |
| 32 | 10099 | 34 | 297 |
| 60 | 17143 | 34 | 504 |
| 61 | 17517 | 34 | 515 |
| 62 | 17499 | 34 | 515 |

误差方面，软硬件运算在六位小数范围内其误差值为 0，可见本设计确实达到了高精度的要求。

6 结束语

IT 产业当今的发展趋势是在更短的开发周期内设计生产出性能更高的产品。SOPC 技术很好的解决了这一问题，它通过软硬件结合设计的方式，让设计者有更大的灵活度在节省硬件资源和提高系统速度之间做出取舍，选择用软件还是硬件实现。本设计以提高处理器精度和速度为目标，通过在 Nios II 处理器中添加用户自定义指令的方式以更多的硬件资源换取更快的计算速度，从而满足现代实时通讯处理的要求。在传统 CORDIC 算法的基础之上，巧妙的改进了算法，达到了更高精度和更快的速度，具有较广泛的应用价值。当然，本设计只完成了正余弦函数的硬件 IP 部分，还有更多的工作等待我们去完成，相信通过不断的努力，我们会完成更多的函数硬件 IP。

参考文献

- [1] .Volder. The CORDIC trigonometric computing technique. IRE Trans. Electronic Computers, 1959, EC-8(3): 334-334.
- [2] . Richard Herveille. Cordic Core Specification . 18, 2001
- [3] 潘宏亮. 浮点指数类超越函数的运算算法研究与硬件实现. 西北工业大学 2006. 3
- [4] IEEE organise. IEEE Standard for Binary Floating-Point Arithmetic
Inc 345 East 47th Street, New York, NY 10017, USA
- [5] .王博立.浮点运算CORDIC之实现与其在3D图形学之应用.台湾国立中山大学 2002.6
- [6] www.Altera.com
- [7] www.opencore.org
- [8] www.edacn.net

原创性声明

我们声明所提交的论文是在导师特别是江国强老师的指导下进行的研究工作及取得的研究成果，并已用于今年的Nios II嵌入式比赛的浮点运算中。尽我们所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果。

李全 陈石平 付佃华

2006-08-19

作者简介：

李 全（1982—）男，桂林电子科技大学通信与信息工程系研究生，主要从事FPGA及ARM嵌入式系统方面的研究。

陈石平（1981—）男，桂林电子科技大学通信与信息工程系研究生，主要从事FPGA通信方面的研究。

付佃华（1982—）男，桂林电子科技大学通信与信息工程系研究生，主要从事微电子方面的研究。

联系方式：

陈石平—地址：桂林电子科技大学研F2#

邮编：541004

电话：13635194211

Email: oldslam@126.com

附录1

```
time1 = alt_timestamp();
time2 = alt_timestamp();
time_overhead = time2 - time1;
printf("time_overhead = %ld\n",time_overhead);

z = theta*AN;

time1 = alt_timestamp();
c1 = FLOAT_SIN(z);
time2 = alt_timestamp();

printf("hard: sin(%f)=%f\n",theta,c1);
printf("hardware use ticks : %ld\n",time2-time1-time_overhead);

w = theta*WR;

time1 = alt_timestamp();
c2 = sin(w);
time2 = alt_timestamp();

printf("soft: sin(%f)=%f\n",theta,c2);
printf("software use ticks : %ld\n",time2-time1-time_overhead);
printf("different: %f\n\n",c2-c1);
theta+=1;
```