CEG5010: The CORDIC Algorithm

Introduction

- COordinate Rotation DIgital Computer
- Efficient method to compute sin, cos, tan, $\sin^{-1}$, $\cos^{-1}$, $\tan^{-1}$, multiplication, division, $\sqrt{}$, sinh, cosh, tanh
  – Only uses shifts, additions and a very small lookup table

2004年4月1日
星期四 (3)

Rotations

Rotating [x y] by ϕ

$$\begin{aligned} x' &= x\cos(\phi) - y\sin(\phi) \\ y' &= y\cos(\phi) + x\sin(\phi). \end{aligned}$$

Rearranging

$$\begin{aligned} x' &= \cos(\phi)(x - y\tan(\phi)) \\ y' &= \cos(\phi)(y + x\tan(\phi)). \end{aligned}$$

2004年4月1日
星期四 (4)

Key idea

$$\begin{aligned} x' &= \cos(\phi)(x - y\tan(\phi)) \\ y' &= \cos(\phi)(y + x\tan(\phi)). \end{aligned}$$

Can compute rotation ϕ in steps where each step is of size

$$\tan(\phi) = \pm 2^{-i}.$$

Iterative rotations

$$
\begin{aligned}
x_{i+1} &= K_i(x_i - (y_i d_i 2^{-i})) \\
y_{i+1} &= K_i(y_i + (x_i d_i 2^{-i})).
\end{aligned}
$$

where $d_i = \pm 1$ and $K_i = \cos(\tan^{-1} 2^{-i})$

Choose d_i so that after n iterations the rotated angle is ϕ

K_i values

$$\cos(\tan^{-1} 2^{-i}) = 1/\sqrt{(1 + 2^{-2i})}.$$

$$K = \prod_{i=1}^{n} \frac{1}{\sqrt{(1 + 2^{-2i})}},$$

As $n \to \infty$, $K \to 0.6073$ (constant factor which needs to be corrected for)

d_i decision (rotation mode)

Z_i is introduced to keep track of the angle that has been rotated (z0 = ϕ)

$$
\begin{aligned}
x_{i+1} &= x_i - (y_i d_i 2^{-i}) \\
y_{i+1} &= y_i + (x_i d_i 2^{-i}) \\
z_{i+1} &= z_i - d_i \tan^{-1}(2^{-i})
\end{aligned}
$$

$$
d_i = \begin{cases} -1 & if\ z_i < 0 \\ +1 & otherwise \end{cases}
$$

After n iterations

$$
\begin{aligned}
x_n &= \frac{1}{K}(x_0 cos(z_0) - y_0 sin(z_0)) \\
y_n &= \frac{1}{K}(y_0 cos(z_0) + x_0 sin(z_0)) \\
z_n &\approx 0.
\end{aligned}
$$

Question: What is the procedure to compute sin and cos?

d_i decision (vectoring mode)

$$d_i = \begin{cases} +1 & if\ y_i < 0 \\ -1 & otherwise. \end{cases}$$

$$\begin{aligned} x_n &= \frac{1}{K}\sqrt(x_0^2 + y_0^2) \\ y_n &\approx 0 \\ z_n &= z_0 + \tan^{-1}(y_0/x_0). \end{aligned}$$

- y_n minimized use to compute $\tan^{-1}$ and magnitude

Linear functions instead of trig

$$\begin{aligned} x_{i+1} &= x_i - 0(y_i d_i 2^{-i}) = x_i \\ y_{i+1} &= y_i + (x_i d_i 2^{-i}) \\ z_{i+1} &= z_i - d_i 2^{-i} \end{aligned}$$

$$d_i = \begin{cases} -1 & if\ z_i < 0 \\ +1 & otherwise. \end{cases}$$

2004年4月1日
星期四 (11)

After n iterations

$$
\begin{aligned}
x_n &= x_0 \\
y_n &= y_0 + x_0 z_0 \\
z_n &= 0,
\end{aligned}
$$

No need for K_n correction.

2004年4月1日
星期四 (12)

Division

$$
d_i = \begin{cases} +1 & if\ y_i < 0 \\ -1 & otherwise. \end{cases}
$$

$$
\begin{aligned}
x_n &= x_0 \\
y_n &= y_0 \\
z_n &= z_0 - y_0/x_0.
\end{aligned}
$$

No need for K_n correction.

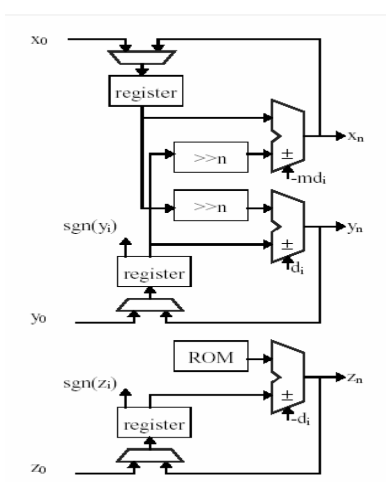
2004年4月1日
星期四 (13)

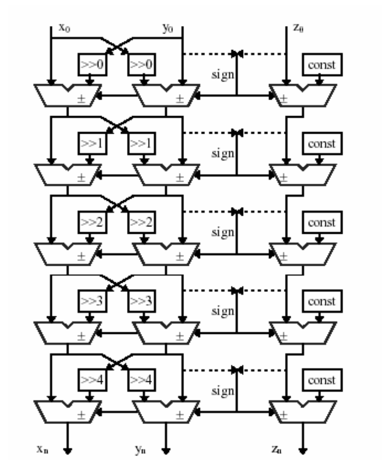
Hyperbolic functions

- Similarly, can get cosh and sinh using $tanh^{-1}$ instead of tan^{-1}
- Can also get ln and exp easily

2004年4月1日
星期四 (14)

Andraka's iterative and unrolled cordic structure





Implementation

- Can develop generalized cordic processors which can compute many different functions using similar hardware
- Implementations can be bit serial and/or pipelined as well

Precision

- Need n iterations for n bits
- Converges for $-99.7 \leq z \leq 99.7$ (sum of all the angles $\tan^{-1}(2^{-i})$, i = 0 .. n)
  - must convert to this range first

Exercise

- Calculate √2/K using the CORDIC algorithm (4 iterations)
- Hint: use vectoring mode

atan(2 ** (-0)) = 0.785398
atan(2 ** (-1)) = 0.463648
atan(2 ** (-2)) = 0.244979
atan(2 ** (-3)) = 0.124355
atan(2 ** (-4)) = 0.062419

Conclusion

- CORDIC algorithms are an efficient method to compute many different functions
- Low area, high speed
- Used in calculators, DSPs, math coprocessors and supercomputers.

References

- Ray Andraka, “A survey of CORDIC algorithms for FPGAs”, FPGA '98. Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays, Feb. 22-24, 1998, Monterey, CA. pp191-200 (http://www.andraka.com/cordic.htm)