EE 5324 – VLSI Design II

Part IX: CORDIC Algorithms

Kia Bazargan

University of Minnesota

References and Copyright

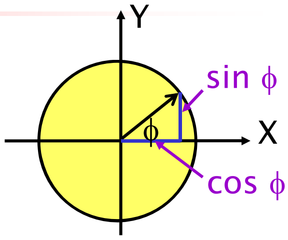
- Textbook referenced
  - [Par00] B. Parhami
    "Computer Arithmetic: Algorithms and Hardware Designs"
    Oxford University Press, 2000.
- Slides used(*Modified by Kia when necessary*)
  - [©Oxford U Press] © Oxford University Press,
    New York, 2000
    Slides for [Par00] (With permission from the author)
    http://www•ece•ucsb•edu/Faculty/Parhami/files_n_docs•htm
  - [©Wilde] © Prof. Doran Wilde, Lecture notes on the Computer Arithmetic Course, ECE Dept., Brigham Young University

What is CORDIC?

- How to evaluate trigonometric functions?
  - Table lookup
  - Polynomial approximations
  - CORDIC
- CORDIC (COordinate Rotation DIgital Computer)
  - Introduced in 1959 by Jack E. Volder
  - Rotate vector (1,0) by ϕ to get (cos ϕ, sin ϕ)
  - Can evaluate many functions
  - Rotation reduced to shift-add operations
  - Convergence method (iterative)
    - N iterations for N-bit accuracy
  - Delay / hardware costs comparable to division or square rooting!

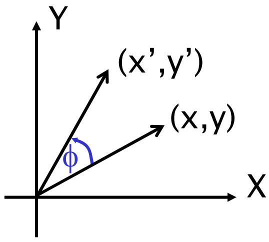
Basic CORDIC Transformations

- Basic idea
  - Rotate (1,0) by ϕ degrees to get (x,y): x=cos(ϕ), y=sin(ϕ)



- Rotation of any (x,y) vector:

$$x' = x.\cos(\phi) - y.\sin(\phi)$$
$$y' = y.\cos(\phi) + x.\sin(\phi)$$



- Rearrange as:

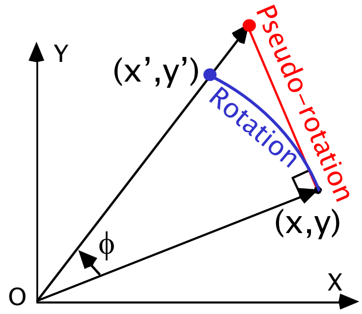
$$x' = \cos(\phi).[x - y.\tan(\phi)]$$
$$y' = \cos(\phi).[y + x.\tan(\phi)]$$

Note: $\frac{\sin(\phi)}{\cos(\phi)} = \tan(\phi)$

Rotation and Magnitude Components

$$x' = \cos(\phi).[x - y.\tan(\phi)]$$
$$y' = \cos(\phi).[y + x.\tan(\phi)]$$

- Two components:
  - *cos(ϕ)*
    - o Reduces the magnitude of the vector
    - o If don't multiply ➔ pseudo rotation
  - *tan(ϕ)*
    - o Rotates the vector
    - o Break ϕ into a series of successively shrinking angles α_i such that: $\tan(\alpha_i) = 2^{-i}$
    - o Should we use all α_i 's?



[© Oxford U Press]

Pre-computation of $\tan(\alpha_i)$

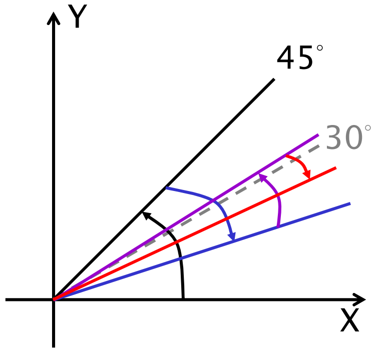
- Find α_i such that $\tan(\alpha_i)=2^{-i}$: (or, $\alpha_i=\tan^{-1}(2^{-i})$)

i	α_i	$\tan(\alpha_i)$	
0	45.0°	1	$= 2^{-0}$
1	26.6°	0.5	$= 2^{-1}$
2	14.0°	0.25	$= 2^{-2}$
3	7.1°	0.125	$= 2^{-3}$
4	3.6°	0.0625	$= 2^{-4}$
5	1.8°	0.03125	$= 2^{-5}$
6	0.9°	0.015625	$= 2^{-6}$
7	0.4°	0.0078125	$= 2^{-7}$
8	0.2°	0.00390625	$= 2^{-8}$
9	0.1°	0.001953125	$= 2^{-9}$

- Note: decreasing α_i.
  - Possible to write any angle $\phi = \pm\alpha_0 \pm \alpha_1 \pm \ldots \pm \alpha_9$ as long as $-99.7^\circ \le \phi \le 99.7^\circ$ (which covers –90..90)
  - Convergence possible: $\alpha_i \le \Sigma^N_{j=i+1}\alpha_{i+1}$ (10^{-5} deg accuracy)

Example: Rewriting Angles in Terms of α_i

- Example: $\phi=30.0^\circ$
  - Start with α_0 = 45.0 (> 30.0)
  - 45.0 – 26.6 = 18.4 (< 30.0)
  - 18.4 + 14.0 = 32.4 (> 30.0)
  - 32.4 – 7.1 = 25.3 (< 30.0)
  - 25.3 + 3.6 = 28.9 (< 30.0)
  - 28.9 + 1.8 = 30.7 (> 30.0)
  - . . .
  - ϕ = 30.0
    ≈ 45.0 – 26.6 + 14.0 – 7.1 + 3.6
    + 1.8 – 0.9 + 0.4 – 0.2 + 0.1
    = 30.1



Rotation Reduction

$$x' = \cos(\phi).[x - y.\tan(\phi)]$$
$$y' = \cos(\phi).[y + x.\tan(\phi)]$$

- Rewrite in terms of α_i: ($0 \le i \le n$)

$$x_{i+1} = \cos(\alpha_i).[x_i - y_i.d_i.\tan(\alpha_i)]$$
$$y_{i+1} = \cos(\alpha_i).[y_i + x_i.d_i.\tan(\alpha_i)]$$
$\rightarrow$
$$\boxed{\begin{aligned} x_{i+1} &= K_i.[x_i - y_i.d_i.2^{-i}] \\ y_{i+1} &= K_i.[y_i + x_i.d_i.2^{-i}] \end{aligned}}$$

- Where:

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

Note: $\cos(\alpha_i) = \cos(-\alpha_i)$

- What about K_i's?

Taking Care of the Magnitude

$$x_{i+1} = K_i.[x_i - y_i.d_i.2^{-i}]$$
$$y_{i+1} = K_i.[y_i + x_i.d_i.2^{-i}]$$

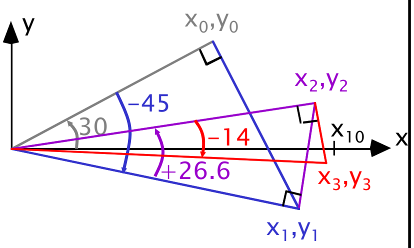
- Observations:
  - We choose to always use ALL α_i terms, with +/- signs
  - $K_i = \cos(\alpha_i) = \cos(-\alpha_i)$
  - At each step, we multiply by $\cos(\alpha_i)$ [constant?]
- Let the multiplications aggregate to:

$$K = \prod_{i=0}^{n} K_i \qquad n \to \infty,\ K = 0.607\,252\,935\ldots$$

  - Multiply this constant ONLY ONCE at the end

Hardware Realization: CORDIC Rotation Mode

- To simplify the hardware:
  - First rotate by ϕ, then rotate by $-d_i.\alpha_i$ to get 0 (no subtraction to compare ϕ and current angle)
- Algorithm: (z is the current angle)
  - Mode: rotation: "at each step, try to make z zero"
  - Initialize x=0.607253,y=0,z=ϕ
  - For i=0 $\to$n
  - d_i = 1 when z>0, else -1
  - $x_{i+1} = x_i - d_i \cdot 2^{-i} \cdot y_i$
  - $y_{i+1} = y_i + d_i \cdot 2^{-i} \cdot x_i$
  - $z_{i+1} = z_i - d_i \cdot \alpha_i$
  - Result: $x_n = \cos(\phi)$, $y_n = \sin(\phi)$
  - Precision: n bits ($\tan^{-1}(2^{-i}) \approx 2^{-i}$)

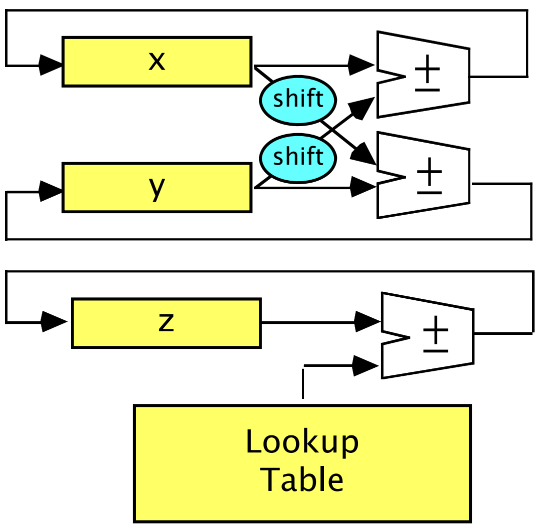


[© Oxford U Press]

CORDIC Rotation Mode C Code

```
// downloaded (and modified by Kia) from
// www.execpc.com/~geezer/embed/cordic.c
#include <stdio.h>
#include <math.h>

#define AG_CONST    0.6072529350
#define FXD(X)         ((long int)((X) * 65536.0))

typedef long int fixed; /* 16.16 fixed-point */

static const fixed Alpha[]={ FXD(45.0),
  FXD(26.565), FXD(14.0362), FXD(7.12502),
  FXD(3.57633), FXD(1.78991), FXD(0.895174),
  FXD(0.447614),FXD(0.223811), FXD(0.111906),
  FXD(0.055953),FXD(0.027977) };

int main(void){
  fixed X, Y, CurrAngle;
  unsigned i;

  X=FXD(AG_CONST);  /* AG_CONST * cos(0) */
  Y=0;                    /* AG_CONST * sin(0) */
  CurrAngle=FXD(28.027);
  for(i=0; i < 12; i++){
    fixed NewX;

    if(CurrAngle > 0) {
      NewX=X - (Y >> i);
      Y+=(X >> i);
      X=NewX;
      CurrAngle -= Alpha[i]; }
    else {
      NewX=X + (Y >> i);
      Y-=(X >> i);
      X=NewX;
      CurrAngle += Alpha[i];
    } // if-else
  } // for (i=...
  printf("cos(28.027)=%6.4f, sin()=%6.4f\n",
          x/65536.0, y/65536.0);
} // main
```


CORDIC Hardware



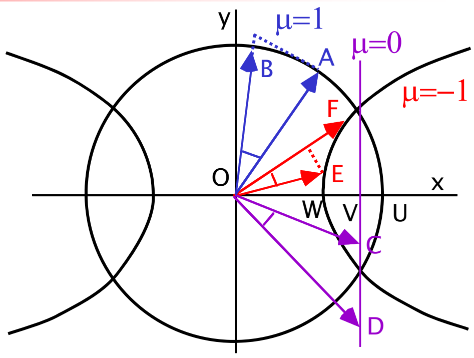
[© Oxford U Press]

CORDIC Vectoring Mode

- Difference with rotation mode?
  - When choosing d_i, instead of trying to make z converge to 0, try to make y_i zero
  - $d_i = -\text{sign}(x_i \cdot y_i)$
- Variables will converge to:
  - $x_n = 1/K\,(x^2 + y^2)^{1/2}$
  - $y_n = 0$
  - $Z_n = z + \tan^{-1}(y/x)$
- Application?
  - If start with x=1, z=0, the final z is $\tan^{-1}(y)$

Generalized CORDIC

- Generalized CORDIC iteration:
  - $x_{i+1} = x_i - \mu \cdot d_i \cdot 2^{-i} \cdot y_i$
  - $y_{i+1} = y_i + d_i \cdot 2^{-i} \cdot x_i$
  - $z_{i+1} = z_i - d_i \cdot e(i)$
- Variations:



μ	Function	e(i)
1	Circular rotation (basic CORDIC)	$\tan^{-1}(2^{-i})$
0	Linear rotation	2^{-i}
−1	Hyperbolic rotation	$\tanh^{-1}(2^{-i})$

[© Oxford U Press]

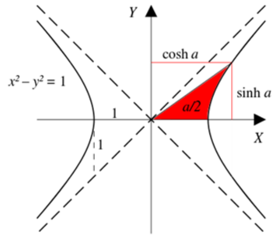
Hyperbolic functions

- Hyperblic sine

$$\sinh x = \frac{e^x - e^{-x}}{2} = \frac{e^{2x}-1}{2e^x} = \frac{1-e^{-2x}}{2e^{-x}}$$

- Hyperbolic cosine

$$\cosh x = \frac{e^x + e^{-x}}{2} = \frac{e^{2x}+1}{2e^x} = \frac{1+e^{-2x}}{2e^{-x}}$$



- Hyperbolic tangent

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x}-1}{e^{2x}+1} = \frac{1-e^{-2x}}{1+e^{-2x}}$$

- Hyperbolic cotangent

$$\coth x = \frac{\cosh x}{\sinh x} = \frac{e^x + e^{-x}}{e^x - e^{-x}} = \frac{e^{2x}+1}{e^{2x}-1} = \frac{1+e^{-2x}}{1-e^{-2x}}$$

Various CORDIC Applications

- Directly computes:
  - sin, cos, sinh, cosh
  - $\tan^{-1}$, $\tanh^{-1}$
  - Division, multiplication
- Also directly computes:
  - $\tan^{-1}(y/x)$
  - y + x.z
  - $(x^2+y^2)^{1/2}$
  - $(x^2-y^2)^{1/2}$
  - $e^z = \sinh(z) + \cosh(z)$

[© Wilde]

Various CORDIC Applications (cont.)

- Indirectly computes:

$$\tan z = \frac{\sin z}{\cos z} \qquad \cos^{-1} w = \tan^{-1} \frac{\sqrt{1-w^2}}{w}$$

$$\tanh z = \frac{\sinh z}{\cosh z} \qquad \sin^{-1} w = \tan^{-1} \frac{w}{\sqrt{1-w^2}}$$

$$\ln w = 2 \tanh^{-1} \left| \frac{w-1}{w+1} \right| \qquad \cosh^{-1} w = \ln\left(w + \sqrt{1-w^2}\right)$$

$$\log_b w = K.\ln w \qquad \sinh^{-1} w = \ln\left(w + \sqrt{1+w^2}\right)$$

$$w^t = e^{t \ln w} \qquad \sqrt{w} = \sqrt{(w+1/4)^2 - (w-1/4)^2}$$

[© Wilde]

Summary of CORDIC Applications

	Rotation mode $d_i = \text{sign}(z_i)$, $z_i \to 0$	Vectoring mode $d_i = -\text{sign}(x_i\, y_i)$, $y_i \to 0$
$\mu = 1$ Circular e(i) = $\tan^{-1} 2^{-i}$	x → CORDIC → K(x cos z − y sin z) y → CORDIC → K(y cos z + x sin z) z → CORDIC → 0 For cos & sin, set x=1/K, y=0 tan z = sin z / cos z	x → CORDIC → $K (x^2 + y^2)^{1/2}$ y → CORDIC → 0 z → CORDIC → $z + \tan^{-1}(y/x)$ For $\tan^{-1}$, set x = 1, z = 0 $\cos^{-1} w = \tan^{-1}[(1-w^2)^{1/2}/w]$ $\sin^{-1} w = \tan^{-1}[w/(1-w^2)^{1/2}]$
$\mu = 0$ Linear e(i)= 2^{-i}	x → CORDIC → x y → CORDIC → y + xz z → CORDIC → 0 For multiplication, set y = 0	x → CORDIC → x y → CORDIC → 0 z → CORDIC → z + y/x For division, set z = 0

Note: in linear mode, limited input range (convergence)

[© Oxford U Press]
[Par00], p. 371

Summary of CORDIC Applications (cont.)

	Rotation mode $d_i=\text{sign}(z_i)$, $z_i \to 0$	Vectoring mode $d_i=-\text{sign}(x_i\, y_i)$, $y_i \to 0$
$\mu = -1$ Hyper-bolic $e(i) = \tanh^{-1} 2^{-i}$	x → CORDIC → $K'(x \cosh z - y \sinh z)$ y → CORDIC → $K'(y \cosh z + x \sinh z)$ z → CORDIC → 0 For cosh & sinh, set $x=1/K'$, $y=0$ $\tanh z = \sinh z / \cosh z$ $\exp(z) = \sinh z + \cosh z$ $w^t = \exp(t \ln w)$	x → CORDIC → $K'(x^2 - y^2)^{1/2}$ y → CORDIC → 0 z → CORDIC → $z + \tanh^{-1}(y/x)$ For $\tanh^{-1}$ set $x = 1$, $z = 0$ $\ln w = 2 \tanh^{-1} \lvert(w - 1)/(w + 1)\rvert$ $w^{1/2} = [(w+1/4)^2 - (w-1/4)^2]^{1/2}$ $\cosh^{-1} w = \ln (w + (1-w^2)^{1/2})$ $\sinh^{-1} w = \ln (w + (1+w^2)^{1/2})$

In the $\mu = -1$ case, steps 4, 13, 40, 121, ... , j , 3j + 1, ... must be repeated for the method to converge. These repetitions are incorporated in the constant K' below.

$x_{i+1} = x_i - \mu \cdot d_i \cdot 2^{-i} \cdot y_i$ $\mu \in \{-1,0,1\}$, $d_i \in \{-1,1\}$

$y_{i+1} = y_i + d_i \cdot 2^{-i} \cdot x_i$ $K = 1.646\,760\,258\,121\ldots$

$z_{i+1} = z_i - d_i \cdot e(i)$ $K' = 0.828\,159\,360\,960\,2\ldots$

[Par00], p. 371
[© Oxford U Press]

To Probe Further…

- Tutorials
  - http://cnmat.cnmat.berkeley.edu/~norbert/cordic/node3.html
  - http://www.execpc.com/~geezer/embed/cordic.htm (including C code)
- Papers
  - Survey paper on FPGA implementation of CORDIC algorithms: http://www.andraka.com/files/crdcsrvy.pdf
  - http://www.taygeta.com/cordic_refs.html
- Hardware implementations
  - http://www.free-ip.com/cordic/
  - http://www.stanford.edu/~chet/cordic.html

Advantages and disadvantages

☺ Simple Shift-and-add Operation.
(2 adders+2 shifters v.s. 4 mul.+2 adder)

⃠ -It needs *n* iterations to obtain *n*-bit precision.
-Slow carry-propagate addition.
-Low throughput rate
-Area consuming shifting operations.

388

How to improve CORDIC ?

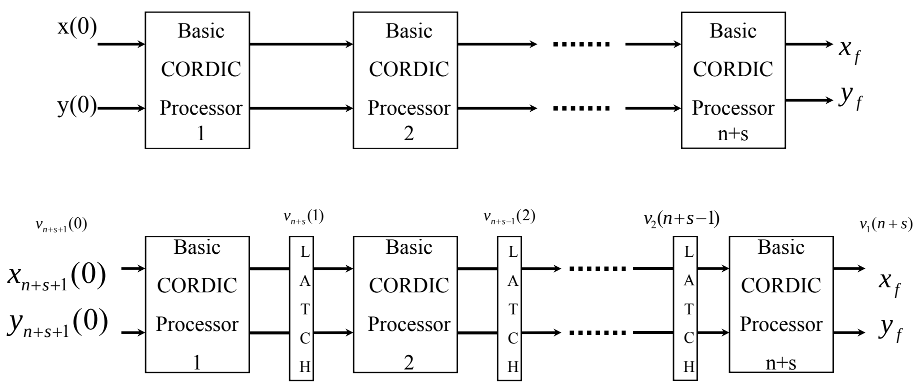
- Use Pipelined Architecture
- Improve the Performance of the Adders (redundant arithmetic, CSA)
- Reduce Iteration Number
  - High radix CORDIC. *(e.g.,* Radix-4, Radix-8)
  - Find a optimized shift sequence *(e.g.,* AR-CORDIC)
- Improve the Scaling Operation
  - Canonical multiplier recoding
  - Force *Km* to 2.

$$\frac{1}{K_m(n)} = \sum_{p=1}^{P} k_p 2^{-i_p}$$

$$k_p = \pm 1$$

389

Parallel and Pipelined Arrays



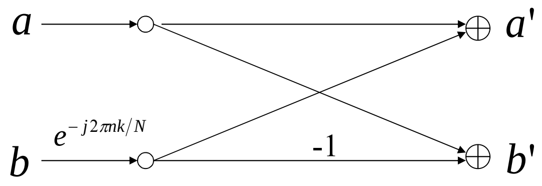
390

Application to DSP Algorithms

- **Linear transformation:**
  - DFT, Chirp-Z transform, DHT, and FFT.
- **Digital filters**:
  - Orthogonal digital filters, and adaptive lattice filters.
- **Matrix based digital signal processing algorithms:**
  - QR factorization, with applications to Kalman filtering
  - Linear system solvers, such as Toeplitz and covariance system solvers,……,*etc.*

391

FFT application

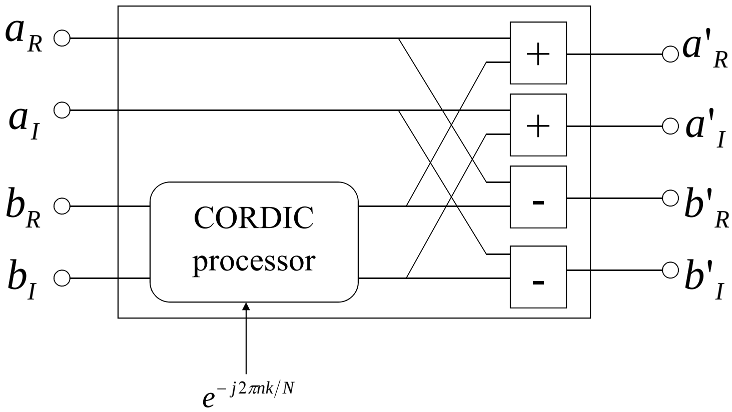


$$a' = a + b \cdot e^{-j2\pi nk/N}$$

$$b' = a - b \cdot e^{-j2\pi nk/N}$$

392

Butterfly unit



393

Conclusions

1. In some cases, CORDIC evaluates rotational functions **more efficiently** than MAC units.
2. CORDIC **saves more hardware cost.**
3. By the **regularity**, the CORDIC based architecture is very suitable for implementation with pipelined VLSI array processors.
4. The utility of the CORDIC based architecture lies in its **generality** and **flexibility.**

394