Lecture Series: Architectural Design of Digital Integrated circuits:

Lec 41) : CORDIC Architecture

CORDIC:

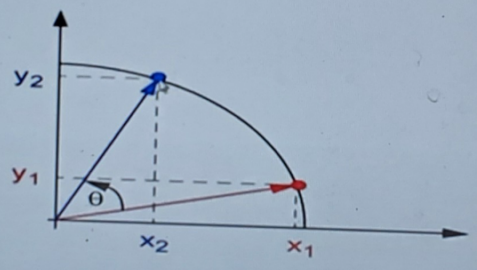
↳ Coordinate Rotation Digital Computer.

Cartesian Coordinate Plane Rotations

Cartesian Coordinate Plane Rotations

- The standard method of rotating a point $((x_1, y_1)$ by θ degrees in the xy plane to a point (x_2, y_2) is given by the well known equations:

$$x_2 = x_1 \cos\theta - y_1 \sin\theta$$
$$y_2 = x_1 \sin\theta + y_1 \cos\theta$$



- This is variously known as a plane rotation, a vector rotation, or in linear (matrix) algebra, a Givens Rotation.

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

→ Rotation in matrix form.

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \cos\theta \begin{pmatrix} 1 & -\tan\theta \\ \tan\theta & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

Pseudo - Rotations :

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 - y_1 \tan\theta \\ x_1 \tan\theta + y_1 \end{pmatrix}$$

→ Rotating without $\cos\theta$.

→ This $\cos\theta$ will be included later on in the algorithm.

→ **NOTE:** $\frac{1}{1+\tan^2\theta} = \frac{1}{1+\frac{\sin^2\theta}{\cos^2\theta}} = \frac{\cos^2\theta}{\sin^2\theta+\cos^2\theta} = \cos^2\theta$

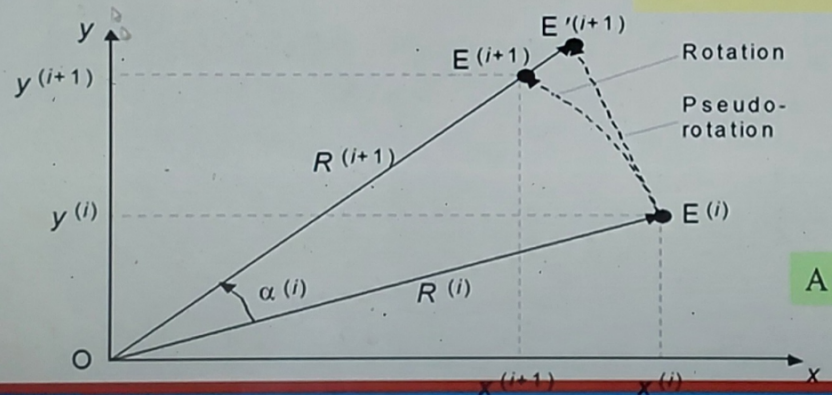
⇒ $\frac{1}{\sqrt{1+\tan^2\theta}} = \cos\theta$

Pseudorotating a Vector $(x^{(i)}, y^{(i)})$ by the Angle $\alpha^{(i)}$

$$x^{(i+1)} = x^{(i)} - y^{(i)} \tan \alpha^{(i)}$$
$$y^{(i+1)} = y^{(i)} + x^{(i)} \tan \alpha^{(i)}$$
$$z^{(i+1)} = z^{(i)} - \alpha^{(i)}$$

Pseudorotation: Whereas a real rotation does not change the length $R(i)$ of the vector, a pseudorotation step increases its length to:

$$R^{(i+1)} = R^{(i)} / \cos a^{(i)} = R^{(i)} (1 + \tan^2 a^{(i)})^{1/2}$$



A pseudorotation step in CORDIC

z^{i+1} → Accumulated angle of rotations

Basic CORDIC Iterations

$$x^{(i+1)} = x^{(i)} - d_i y^{(i)} 2^{-i}$$
$$y^{(i+1)} = y^{(i)} + d_i x^{(i)} 2^{-i}$$
$$z^{(i+1)} = z^{(i)} - d_i \tan^{-1} 2^{-i} = z^{(i)} - d_i e^{(i)}$$

i	$e^{(i)}$ in degrees (approximate)	$e^{(i)}$ in radians (precise)
0	45.0	0.785 398 163
1	26.6	0.463 647 609
2	14.0	0.244 978 663
3	7.1	0.124 354 994
4	3.6	0.062 418 810
5	1.8	0.031 239 833
6	0.9	0.015 623 728
7	0.4	0.007 812 341
8	0.2	0.003 906 230
9	0.1	0.001 953 123

Value of the function $e^{(i)} = \tan^{-1} 2^{-i}$, in degrees and radians, for $0 \le i \le 9$

Example: 30° angle

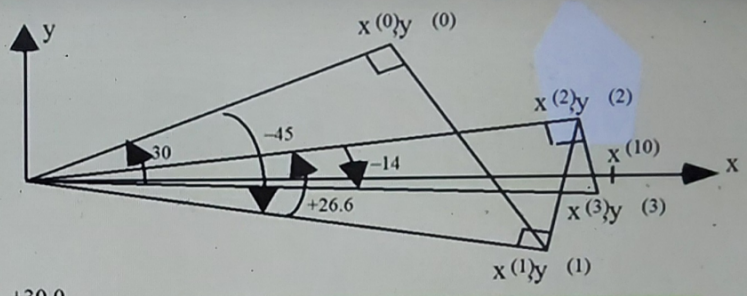
$30.0 \cong 45.0 - 26.6 + 14.0 - 7.1 + 3.6 + 1.8 - 0.9 + 0.4 - 0.2 + 0.1 = 30.1$

Lec 42) : CORDIC Architecture

Choosing the Angles to Force z to Zero

$$x^{(i+1)} = x^{(i)} - d_i y^{(i)} 2^{-i}$$
$$y^{(i+1)} = y^{(i)} + d_i x^{(i)} 2^{-i}$$
$$z^{(i+1)} = z^{(i)} - d_i \tan^{-1} 2^{-i}$$
$$= z^{(i)} - d_i e^{(i)}$$

i	$z^{(i)}$		$d_i e^{(i)}$		$z^{(i+1)}$
0	+30.0	−	45.0	=	−15.0
1	−15.0	+	26.6	=	+11.6
2	+11.6	−	14.0	=	−2.4
3	−2.4	+	7.1	=	+4.7
4	+4.7	−	3.6	=	+1.1
5	+1.1	−	1.8	=	−0.7
6	−0.7	+	0.9	=	+0.2
7	+0.2	−	0.4	=	−0.2
8	−0.2	+	0.2	=	+0.0
9	+0.0	−	0.1	=	−0.1



+30.0

The first three of 10 pseudorotations leading from $(x^{(0)}, y^{(0)})$ to $(x^{(10)}, 0)$ in rotating by +30°.

Choosing the signs of the rotation angles in order to force z to 0

d_i → direction of rotation

z^{i+1} → Accumulate angle of rotation.

→ Choosing the signs of the rotation angles in order to force z to 0.

Angle Accumulator

- The pseudo rotation shown earlier can now be expressed for each iteration as:

$$x^{(i+1)} = x^{(i)} - d_i(2^{-i} y^{(i)})$$
$$y^{(i+1)} = y^{(i)} + d_i(2^{-i} x^{(i)})$$

- At this stage we introduce a 3rd equation called the Angle Accumulator which is used to keep track of the accumulative angle rotated at each iteration:

$$z^{(i+1)} = z^{(i)} - d_i \theta^{(i)} \quad \text{(Angle Accumulator)}$$

where $d_i = +/- 1$

- The symbol d_i is a decision operator and is used to decide which direction to rotate.

Shift-Add Algorithm

- Hence, the original algorithm has now been reduced to an iterative shift-add algorithm for pseudo-rotations of a vector:

$$x^{(i+1)} = (x^{(i)} - d_i(2^{-i}y^{(i)}))$$
$$y^{(i+1)} = (y^{(i)} + d_i(2^{-i}x^{(i)}))$$
$$z^{(i+1)} = z^{(i)} - d_i\theta^{(i)}$$

- Thus each iteration requires:
  - 2 shifts
  - 1 table lookup ($\theta^{(i)}$ values)
  - 3 additions

Lec 43 : CORDIC Architecture

→ Scaling factor can be precomputed if the number of iterations is known to us and can later be included in the algorithm as system gain.

To simplify the Givens rotation we removed the $\cos\theta$ term to allow us to perform pseudo-rotations. However, this simplification has a **side-effect**. The output values $x^{(n)}$ and $y^{(n)}$ are scaled by a factor K_n known as the Scaling Factor where:

$$K_n = \prod_n 1/(\cos\theta^{(i)}) = \prod_n \left(\sqrt{1+\tan^2\theta^{(i)}}\right) = \prod_n \left(\sqrt{1+2^{(-2i)}}\right)$$

$$K_n \to 1.6476 \text{ as } n \to \infty$$

$$1/K_n \to 0.6073 \text{ as } n \to \infty$$

$$n = \text{number of iterations}$$

However, if we know the number of iterations that will be performed then we can precompute the value of $1/K_n$ and correct the final values of $x^{(n)}$ and $y^{(n)}$ by multiplying them by this value.

Mode of operation of CORDIC

1). Rotation Mode
2). Vectoring Mode

→ Mode of operation denotes the value of d_i

Rotation Mode :

Lecture 43: CORDIC Architecture

Rotation Mode

- The CORDIC method is operated in one of two modes;
- The mode of operation dictates the condition for the control operator d_i;
- In Rotation Mode choose: $d_i = \text{sign}(z^{(i)}) \Rightarrow z^{(i)} \rightarrow 0$;
- After n iterations we have:

$$x^{(n)} = K_n(x^{(0)}\cos z^{(0)} - y^{(0)}\sin z^{(0)})$$
$$y^{(n)} = K_n(y^{(0)}\cos z^{(0)} + x^{(0)}\sin z^{(0)})$$
$$z^{(n)} = 0$$

→ Final Result

- Can compute $\cos z^{(0)}$ and $\sin z^{(0)}$ by starting with $x^{(0)} = 1/K_n$ and $y^{(0)} = 0$

Vectoring Mode :

Vectoring Mode

- In Vectoring Mode choose: $d_i = -\text{sign}(x^{(i)}y^{(i)}) \Rightarrow y^{(i)} \rightarrow 0$
- After n iterations we have:

$$x^{(n)} = K_n\left(\sqrt{(x^{(0)})^2 + (y^{(0)})^2}\right)$$
$$y^{(n)} = 0$$
$$z^{(n)} = z^{(0)} + \tan^{-1}\left(\frac{y^{(0)}}{x^{(0)}}\right)$$

- Can compute $\tan^{-1} y^{(0)}$ by setting $x^{(0)} = 1$ and $z^{(0)} = 0$

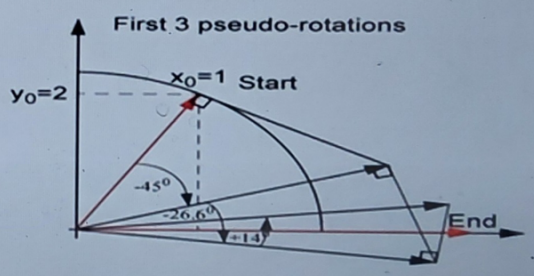
In Vectoring Mode the decision operator d_i obeys the condition:

$$d_i = -\text{sign}(x^{(i)}y^{(i)})$$

Thus, we input $x^{(0)}$ and $y^{(0)}$ ($z^{(0)} = 0$) and then drive $y^{(0)}$ towards 0.

Example: calculate $\tan^{-1}(y^{(0)}/x^{(0)})$ where $y^{(0)} = 2$ and $x^{(0)} = 1$

i	$z^{(i)}$	θ^i	$y^{(i)}$
0	0	45	2
1	45	26.6	1
2	71.6	14	-0.5
3	57.6	7.1	0.375
4	64.7	3.6	-0.078
5	61.1	1.8	0.151
6	62.9	0.9	0.039
7	63.8	0.4	-0.019
8	63.4	0.2	0.009



Example of computing $\tan^{-1}\left(\frac{y}{x}\right)$ using CORDIC in Vectoring Mode.

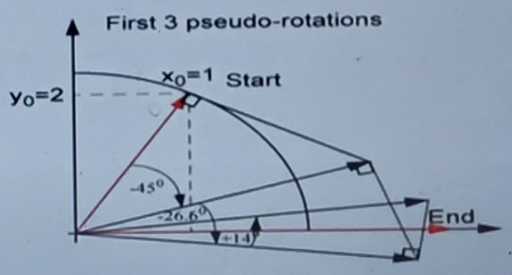
In Vectoring Mode the decision operator d_i obeys the condition:

$$d_i = -sign(x^{(i)}y^{(i)})$$

Thus, we input $x^{(0)}$ and $y^{(0)}$ ($z^{(0)} = 0$) and then drive $y^{(0)}$ towards 0.

Example: calculate $\tan^{-1}(y^{(0)}/x^{(0)})$ where $y^{(0)} = 2$ and $x^{(0)} = 1$

i	$z^{(i)}$	θ^i	$y^{(i)}$
0	0	45	2
1	45	26.6	1
2	71.6	14	-0.5
3	57.6	7.1	0.375
4	64.7	3.6	-0.078
5	61.1	1.8	0.151
6	62.9	0.9	0.039
7	63.8	0.4	-0.019
8	63.4	0.2	0.009



→ Only 'd_i' changes when we go from CORDIC in rotation mode to vectoring mode. The steps mentioned in the Shift-Add algorithm mentioned in pg 2 remains the same for CORDIC in vectoring mode and rotation mode.

Summary of CORDIC in Circular Coordinate System:

Circular Coordinate System

- So far only pseudo-rotations in a Circular Coordinate System have been considered.
- Thus, the following functions can be computed:

Coordinate System	Rotation Mode $z^{(i)} \rightarrow 0$; $d_i = sign(z^{(i)})$	Vectoring Mode $y^{(i)} \rightarrow 0$; $d_i = -sign(x^{(i)}y^{(i)})$
Circular	x, y, z → CORDIC → $K(x\cos z - y\sin z)$, $K(y\cos z + x\sin z)$, 0 For cos z & sin z, set x = 1/K, y = 0	x, y, z → CORDIC → $K(x^2+y^2)^{1/2}$, 0, $z + \tan^{-1}(y/x)$ For $\tan^{-1}$ y, set x = 1, z = 0

- However, more functions can be computed if we use other coordinate systems.

Other coordinate Systems

The advantage of using other coordinate systems with the CORDIC algorithm is that it allows more functions to be calculated. The drawback is that the system becomes more complex. The set of rotation angles used for the Circular Coordinate System are no longer valid when using the CORDIC algorithm with a Linear or Hyperbolic system. Hence, two other sets of angles are used for rotations made in these systems.

We shall see shortly that the CORDIC equations can be generalised for the 3 coordinate systems and that this involves the introduction of two new variables to the equations. One of these new variables ($e^{(i)}$) represents the set of angles used to represent rotations in the appropriate coordinate system.

Generalized CORDIC Equations

Generalized CORDIC Equations

- With the addition of two other Coordinate Systems the CORDIC equations can now be generalised to accommodate all three systems:

$$x^{(i+1)} = (x^{(i)} - \mu d_i(2^{-i}y^{(i)}))$$
$$y^{(i+1)} = (y^{(i)} + d_i(2^{-i}x^{(i)}))$$
$$z^{(i+1)} = z^{(i)} - d_i e^{(i)}$$

- Circular Rotations: $\mu = 1, e^{(i)} = \tan^{-1}2^{-i}$
- Linear Rotations: $\mu = 0, e^{(i)} = 2^{-i}$
- Hyperbolic Rotations: $\mu = -1, e^{(i)} = \tanh^{-1}2^{-i}$

Summary of CORDIC Functions

	Rotation Mode: d_i=sign($z^{(i)}$); $z^{(i)} \rightarrow 0$	Vectoring Mode: d_i=-sign($x^{(i)}y^{(i)}$); $y^{(i)} \rightarrow 0$
Circular $\mu = 1$ $e^{(i)} = \tan^{-1}2^{-i}$	x → K(x.cos z - y.sin z) y → K(y.cos z + x.sin z) z → 0 For cos z & sin z, set x = 1/K, y = 0	x → K($x^2 + y^2$)$^{1/2}$ y → 0 z → z + $\tan^{-1}$(y/x) For $\tan^{-1}$ y, set x = 1, z = 0
Linear $\mu = 0$ $e^{(i)} = 2^{-i}$	x → x y → y + (x.z) z → 0 For multiplication, set y = 0	x → x y → 0 z → z + (y/x) For division, set z = 0
Hyperbolic $\mu = -1$ $e^{(i)} = \tanh^{-1}2^{-i}$	x → K*(x.cosh z - y.sinh z) y → K*(y.cosh z + x.sinh z) z → 0 For cosh z & sinh z, set x = 1/K*, y = 0	x → K*($x^2 - y^2$)$^{1/2}$ y → 0 z → z + $\tanh^{-1}$(y/x) For $\tanh^{-1}$y, set x = 1, z = 0

Scaling factor used in Hyperbolic and Circular Rotations

When using the CORDIC algorithm for Hyperbolic rotations the scaling factor K is different from the one used for Circular rotations.

The Hyperbolic scaling factor is denoted K^* and is calculated using the equation:

$$K^*_n = \prod_n \left(\sqrt{1 - 2^{(-2i)}}\right)$$

$K^*_n \to 0.82816$ as $n \to \infty$

$1/K^*_n \to 1.20750$ as $n \to \infty$

n = number of iterations

→ This is the scaling factor that we will be using in hyperbolic rotation (Analogous to 0.6072 in Circular Rotation)

Other functions that can be computed using CORDIC

Other Functions

- Although the CORDIC algorithms can only compute a limited number of functions directly, there are many more that can be achieved indirectly:

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

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

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

$$e^z = \sinh z + \cosh z \qquad \sinh^{-1} w = \ln(w + \sqrt{1 + w^2})$$

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

Precision & Convergence

- For k bits of precision in trigonometric functions, k iterations are required.
- Convergence is guaranteed for Circular & Linear CORDIC using angles in range $-99.7 \leq z \leq 99.7$:
  - for angles outside this range use standard trig identities.

→ when angle does not lie in $-90° \leq \theta \leq 90°$

- Elemental rotations using Hyperbolic CORDIC do not converge:
  - convergence is achieved if certain iterations are repeated;
  - $i = 4, 13, 40, \ldots\ldots, k, 3k+1, \ldots$; $k = 1, 2, 3 \ldots$