Topic 5

Functions Evaluation

Peter Cheung
Department of Electrical & Electronic Engineering
Imperial College London

URL: www.ee.imperial.ac.uk/pcheung/
E-mail: p.cheung@imperial.ac.uk

About this Topic

- Sine/Cosine functions generation methods
- Functions generation using polynomial approximation
- Distributed arithmetic
  - Constant coefficient filters
  - Inner-product computation

Sine/Cosine Generation

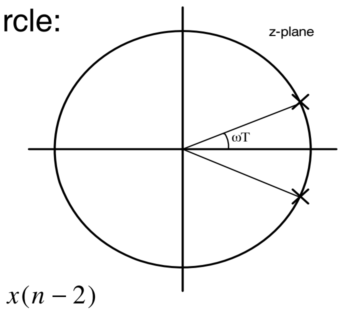
- Sine and cosine functions - very common in communications and DSP applications
  - e.g. modulation, demodulation, FFT, spectral analysis
- We will consider this as an example of system level architecture
- 4 Methods are considered:-
  - Recursive evaluation
  - Direct Table Lookup
  - Two-level table lookup
  - CORDIC algorithm

Method 1: Recursive Evaluation

- Basic idea: place pole pair on unit circle:

$$H(z) = \frac{1}{(z - e^{j\omega T}) \bullet (z - e^{-j\omega T})}$$

$$= \frac{z^{-2}}{(1 - 2\cos \omega T z^{-1} + z^{-2})}$$



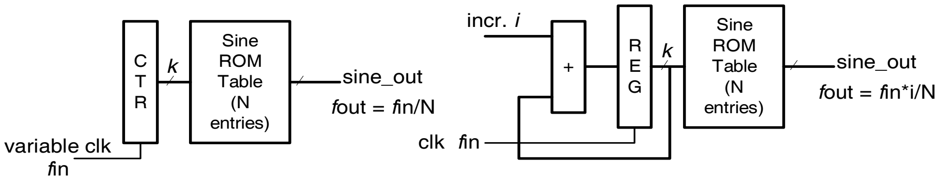
- Rewrite as difference equation:

$$y(n) = 2\cos \omega T \bullet y(n-1) - y(n-2) + x(n-2)$$

- This will oscillate at frequency ω with x(n-2) = 0
- Limitations:
  - Fixed frequency only
  - Amplitude may grow or decay - sensitive to quantization noise
  - No quadrature signal (i.e. cosine and sine together)

Method 2: Direct Table Lookup

- ◆ Store one cycle of sine wave in ROM lookup table
- ◆ Two approaches to change output frequency:
  - 1. Use address counter with variable clock frequency
  - 2. Use address adder with fixed clock frequency



- ◆ Maximum clock frequency limited by access time of ROM.
- ◆ Exploit symmetry of sine wave and store one quadrant
  - reduce size of ROM by a factor of 4

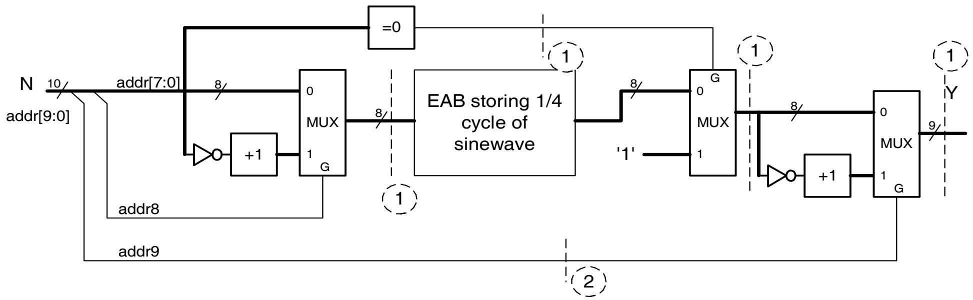
Method 2: Direct Table Lookup (Example)

- ◆ Example: Use embedded block RAM (EAB) in 256 x 8 bit configuration to store ¼ cycle of a sine table such that:
  - Mem[K] = 255 * sin (π * K / 512) for K = 0 to 255.
  - Generate the other quadrants by manipulating the address and negating the ROM/RAM values.
  - The rule to generate the EAB address '**reflection**' and amplitude negation are:-

addr9	addr8	Address to EAB	Negation
0	0	addr[7:0]	No
0	1	256 – addr[7:0]	No
1	0	addr[7:0]	Yes
1	1	256 – addr[7:0]	Yes

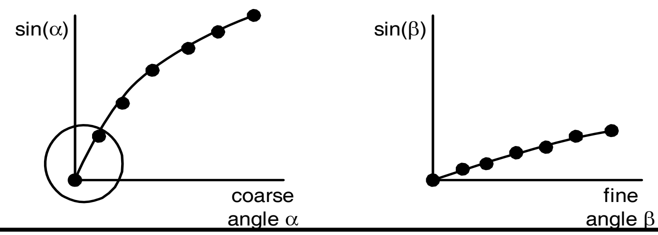
Method 2: Direct Table Lookup (example)

- ◆ This works except for N=256 and 768 when addr[7:0] = 0.
- ◆ Therefore, detect this condition and force output to either +255 or –255.
- ◆ Improve speed by inserting pipeline registers at dotted lines.
- ◆ Numbers in circle indicate number of pipeline register stages.



Method 3: Two level Table Lookup

- ◆ Previous method still requires table of size N/4
- ◆ For fine angular increment, needs very large table
- ◆ Can trade-off computational block for ROM size by using two tables:
  - 1. Coarse angle table
    - ↗ storing sin(α), where $\alpha = \pi k/(2*M)$, for k = 0 to M-1
  - 2. Fine angle table
    - ↗ storing sin(β), where $\beta = \pi k/(2*M*N)$, for k = 0 to N-1



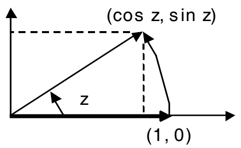
Method 3: Two level Table Lookup (con't)

- Now, compute
  - $\sin(\alpha + \beta) = \sin\alpha \cos\beta + \cos\alpha \sin\beta$
- Requires two multiplies and one add
- Angular resolution now improved to $\pi/(2*M*N)$, or $4*M*N$ angles in one cycle
- Further simplification if M is large and $\beta \approx 0$, then
  - $\sin(\alpha + \beta) \approx \sin\alpha + \beta\cos\alpha \approx \sin\alpha + \beta\sin(90 - \alpha)$
  - No need to have the fine angle table
  - Requires only one multiply
  - Introduces distortion

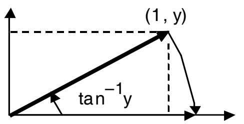
Method 4: Cordic Algorithms

- CORDIC stands for: **CO**ordinate **R**otation **DI**gital **C**omputer
- Invented in 1959 by Jack E. Volder
- Based on the observation :
  - Rotate a unit-length vector (1,0) by an angle z
  - New vector will be at (cos z, sin z)
- Extended by J.S. Walther in 1971 to compute many functions of interest
- Used in virtually all scientific calculators to compute trigonometric functions!

Rotations and Pseudorotations



start at (1, 0)
rotate by z
get cos z, sin z



start at (1, y)
rotate until y = 0
rotation amount is $\tan^{-1}y$

Key ideas in CORDIC

COordinate **R**otation **DI**gital **C**omputer used this method in 1950s; modern electronic calculators also use it

If we have a computationally efficient way of rotating a vector, we can evaluate cos, sin, and $\tan^{-1}$ functions

Rotation by an arbitrary angle is difficult, so use two tricks:

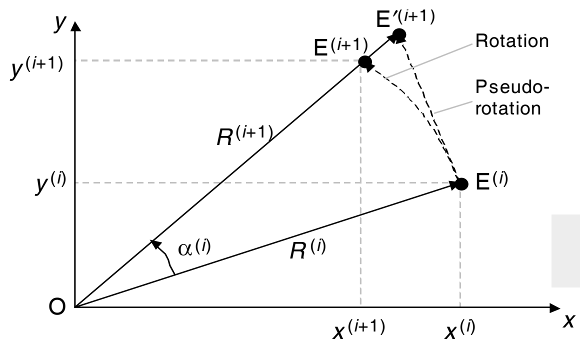
1. Perform **psuedorotations** that require simpler operations
2. Make up the desired angle z from a **set of special angles**
   $z = \alpha^{(1)} + \alpha^{(2)} + \ldots + \alpha^{(m)}$

Source: Parhami

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

$$x^{(i+1)} = x^{(i)}\cos\alpha^{(i)} - y^{(i)}\sin\alpha^{(i)} = (x^{(i)} - y^{(i)}\tan\alpha^{(i)}) / (1 + \tan^2\alpha^{(i)})^{1/2}$$
$$y^{(i+1)} = y^{(i)}\cos\alpha^{(i)} + x^{(i)}\sin\alpha^{(i)} = (y^{(i)} + x^{(i)}\tan\alpha^{(i)}) / (1 + \tan^2\alpha^{(i)})^{1/2}$$
$$z^{(i+1)} = z^{(i)} - \alpha^{(i)}$$

Recall that $\cos\theta = 1/(1 + \tan^2\theta)^{1/2}$



Our strategy: Eliminate the terms $(1 + \tan^2\alpha^{(i)})^{1/2}$ and choose the angles $\alpha^{(i)}$ so that $\tan\alpha^{(i)}$ is a power of 2; need two shift-adds

A pseudorotation step in CORDIC

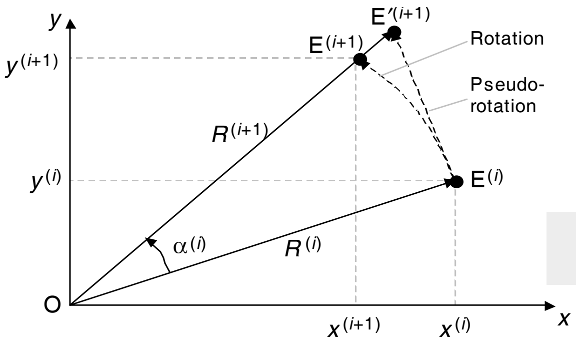
Source: Parhami

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:

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



A pseudorotation step in CORDIC

Source: Parhami

A Sequence of Rotations or Pseudorotations

$$x^{(m)} = x \cos(\Sigma\alpha^{(i)}) - y \sin(\Sigma\alpha^{(i)})$$
$$x^{(m)} = y \cos(\Sigma\alpha^{(i)}) + x \sin(\Sigma\alpha^{(i)})$$
$$z^{(m)} = z - (\Sigma\alpha^{(i)})$$

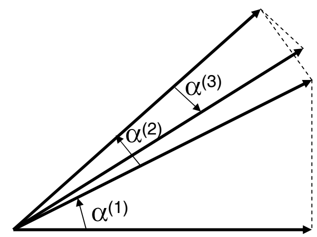
After m real rotations by $\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(m)}$, given $x^{(0)} = x$, $y^{(0)} = y$, and $z^{(0)} = z$

$$x^{(m)} = K(x \cos(\Sigma\alpha^{(i)}) - y \sin(\Sigma\alpha^{(i)}))$$
$$y^{(m)} = K(y \cos(\Sigma\alpha^{(i)}) + x \sin(\Sigma\alpha^{(i)}))$$
$$z^{(m)} = z - (\Sigma\alpha^{(i)})$$

After m pseudorotations by $\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(m)}$, given $x^{(0)} = x$, $y^{(0)} = y$, and $z^{(0)} = z$

where $K = \prod(1 + \tan^2 \alpha^{(i)})^{1/2}$ is a constant if angles of rotation are always the same, differing only in sign or direction

Question: Can we find a set of angles so that any angle can be synthesized from all of them with appropriate signs?



Basic CORDIC Iterations

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

CORDIC iteration: In step i, we pseudorotate by an angle whose tangent is $d_i 2^{-i}$ (the angle $e^{(i)}$ is fixed, only direction d_i is to be picked)

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

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

Source: Parhami

Basic CORDIC iterations

- We can avoid any multiplication by choosing fixed rotation angles $\pm\alpha_i$ such that:

$$\tan \alpha_i = 2^{-i}$$
$$\alpha_i = \tan^{-1} 2^{-i}$$

- Only need shifts instead of multiplications.

i	α_I	$\tan \alpha_i = 2^{-i}$
0	45.000	1.000
1	26.565	0.500
2	14.036	0.250
3	7.125	0.125
4	3.576	0.0625
5	1.790	0.03125

CORDIC rotation

$$x_{i+1} = x_i - y_i \tan \alpha_i$$
$$y_{i+1} = y_i + x_i \tan \alpha_i$$
$$z_{i+1} = z_i - \alpha_i$$

$$\alpha_i = \tan^{-1} 2^{-i}$$
$$\tan \alpha_i = 2^{-i}$$

$$x_{i+1} = x_i - d_i \; y_i 2^{-i}$$
$$y_{i+1} = y_i + d_i \; x_i 2^{-i}$$
$$z_{i+1} = z_i - d_i \; \alpha_i$$

$d_i \in \{-1, 1\}$ as determined by some criterion

CORDIC Iteration complexity

- Each CORDIC rotation requires:
  - 2 shift operations
  - 1 table lookup to find α_i
  - 3 additions
- By rotating by the same set of angles from table (with + or - signs), the scaling factor K can be pre-calculated and stored in another table.

Choosing the Angles to Force *z* to Zero

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

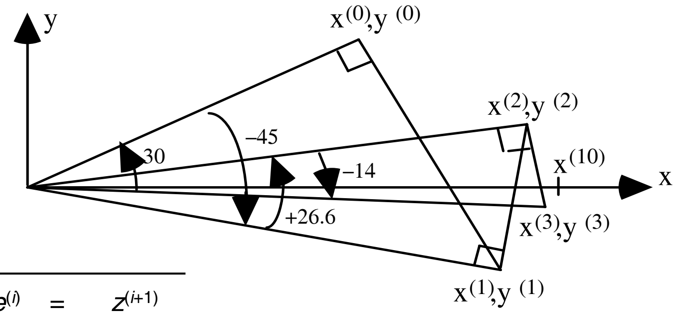
$$x^{(i+1)} = x^{(i)} - d_i \, y^{(i)} 2^{-i}$$
$$y^{(i+1)} = y^{(i)} + d_i \, x^{(i)} 2^{-i}$$
$$z^{(i+1)} = z^{(i)} - d_i \tan^{-1} 2^{-i}$$
$$= z^{(i)} - d_i \, e^{(i)}$$

i	$z^{(i)}$	–	$d_i\, e^{(i)}$	=	$z^{(i+1)}$
					+30.0
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

Source: Parhami

Geometric interpretation (first 3 rotations)

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



i	$z^{(i)}$	–	$d_i\, e^{(i)}$	=	$z^{(i+1)}$
					+30.0
0	+30.0	–	45.0	=	–15.0
1	–15.0	+	26.6	=	+11.6
2	+11.6	–	14.0	=	–2.4
3	…………				

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

Source: Parhami

Why Any Angle Can Be Formed from Our List

Analogy: Paying a certain amount while using all currency denominations (in positive or negative direction) exactly once; red values are fictitious.

\$20 \$10 \$5 \$3 \$2 \$1 \$.50 \$.25 \$.20 \$.10 \$.05 \$.03 \$.02 \$.01

Example: Pay \$12.50

\$20 – \$10 + \$5 – \$3 + \$2 – \$1 – \$.50 + \$.25 – \$.20 – \$.10 + \$.05 + \$.03 – \$.02 – \$.01

Convergence is possible as long as each denomination is no greater than the sum of all denominations that follow it.

Domain of convergence: –\$42.16 to +\$42.16

We can guarantee convergence with actual denominations if we allow multiple steps at some values:

\$20 \$10 \$5 \$2 \$2 \$1 \$.50 \$.25 \$.10 \$.10 \$.05 \$.01 \$.01 \$.01 \$.01

Example: Pay \$12.50

\$20 – \$10 + \$5 – \$2 – \$2 + \$1 + \$.50+\$.25–\$.10–\$.10–\$.05+\$.01–\$.01+ \$.01–\$.01

It can be shown that in hyperbolic CORDIC, convergence is guaranteed only if certain “angles” are used twice.

Source: Parhami

Using CORDIC in Rotation Mode

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

Make z converge to 0 by choosing $d_i = \text{sign}(z^{(i)})$

$$x^{(m)} = \cancel{K}(\cancel{x} \cos z - \cancel{y \sin z}) \quad 0$$
$$y^{(m)} = \cancel{K}(\cancel{y \cos z} + \cancel{x} \sin z) \quad 0$$
$$z^{(m)} = 0$$

where K = 1.646 760 258 121 . . .

For k bits of precision in results, k CORDIC iterations are needed, because $\tan^{-1} 2^{-i} \cong 2^{-i}$ for large i

Start with
$x = 1/K$ = 0.607 252 935 . . .
and $y = 0$
to find cos z and sin z

Convergence of z to 0 is possible because each of the angles in our list is more than half the previous one or, equivalently, each is less than the sum of all the angles that follow it

Domain of convergence is $-99.7^\circ \le z \le 99.7^\circ$, where 99.7° is the sum of all the angles in our list; the domain contains $[-\pi/2, \pi/2]$ radians

Source: Parhami

Compute Sine and Cosine using CORDIC

- Initialise:
  - z = z
  - x = 1/K = 0.607252935…..
  - y = 0
- Iterate with $d_i = \text{sign}(z_i)$
- After m rotations,

$$x_m \approx \cos(z)$$
$$y_m \approx \sin(z)$$
$$z_m \approx 0$$
$$y / x \approx \tan(z)$$

Using CORDIC in Vectoring Mode

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

Make y converge to 0 by choosing $d_i = -\text{sign}(x^{(i)} y^{(i)})$

$$x^{(m)} = K(x^2 + y^2)^{1/2}$$
$$y^{(m)} = 0$$
$$z^{(m)} = \cancel{z} + \tan^{-1}(y/\cancel{x}) \quad 0$$

where K = 1.646 760 258 121 . . .

For k bits of precision in results, k CORDIC iterations are needed, because $\tan^{-1} 2^{-i} \cong 2^{-i}$ for large i

Start with
$x = 1$ and $z = 0$
to find $\tan^{-1} y$

Even though the computation above always converges, one can use the relationship $\tan^{-1}(1/y) = \pi/2 - \tan^{-1} y$ to limit the range of fixed-point numbers encountered

Other trig functions: tan z obtained from sin z and cos z via division; inverse sine and cosine ($\sin^{-1} z$ and $\cos^{-1} z$) discussed later

CORDIC in Vector Mode

- Initialise: z = z, x = x, y = y ☺
- Iterate with $d_i = -\text{sign}(x_i\, y_i)$, which forces y_m towards 0
- After m rotations,

$$x_m = K\ (x^2 + y^2)^{1/2}$$
$$y_m = 0$$
$$z_m = z + \tan^{-1}(y/x)$$
$$K = 1.6467602581\ 21......$$

Use CORDIC to compute arctan(y)

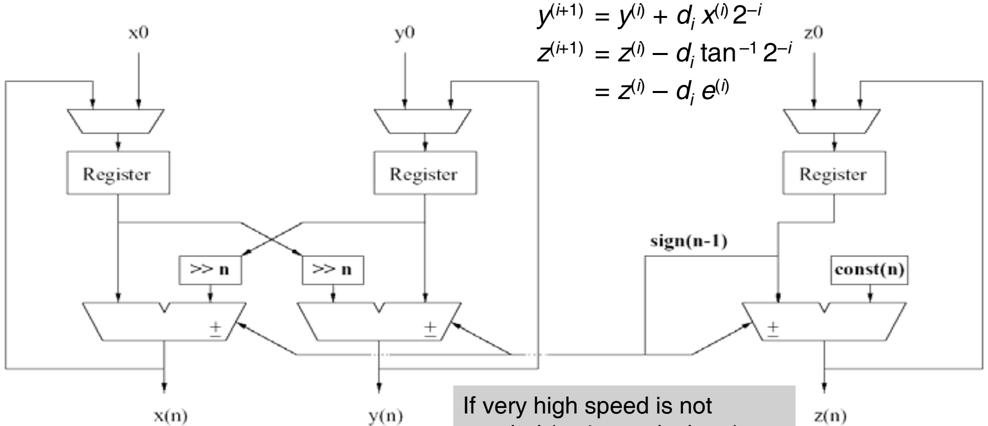
- Initialise: ☺
  - z = 0
  - x = 1
  - y = y
- Iterate with $d_i = -\text{sign}(x_i\, y_i) = -\text{sign}(y_i)$
- After m rotations,

$$z_m = \tan^{-1}(y)$$

- Use identity: $\tan^{-1}(1/y) = \pi/2 - \tan^{-1}y$ to limit range of numbers to manageable size

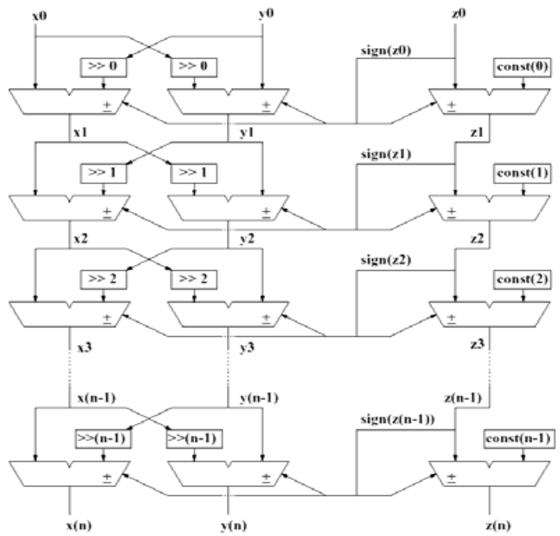
Bit-parallel iterative CORDIC

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

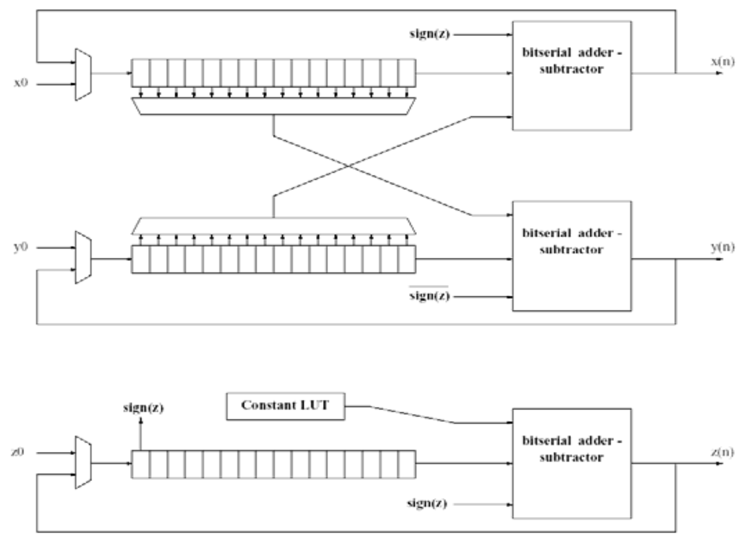


If very high speed is not needed (as in a calculator), a single adder and one shifter would suffice

Bit-parallel unrolled CORDIC



Bit-serial CORDIC



Practical issues

- For k bits precision at output, only k iterations needed.
- For large value of i, $\tan(2^{-i}) \approx 2^{-I}$
- Convergence is guaranteed for angles in range:
  - $-99.7 \le z \le 99.7$ (99.7 being the sum of all angles in table)
- For angles outside this range, use trigonometric rules to convert angle in range.

Generalized CORDIC

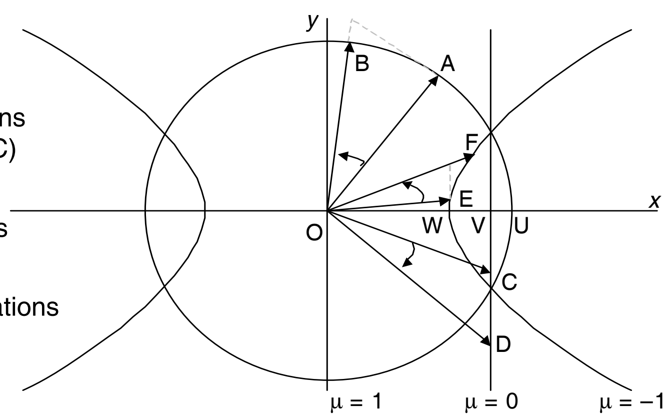
☺

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

$\mu = 1$ Circular rotations (basic CORDIC) $e^{(i)} = \tan^{-1} 2^{-i}$

$\mu = 0$ Linear rotations $e^{(i)} = 2^{-i}$

$\mu = -1$ Hyperbolic rotations $e^{(i)} = \tanh^{-1} 2^{-i}$



Circular, linear, and hyperbolic CORDIC.

Source: Parhami

Universal CORDIC

☺

Directly computes:	Also directly computes:
sin	$\tan^{-1}(y/x)$
cos	$y + xz$
$\tan^{-1}$	$\sqrt{x^2 + y^2}$
sinh	$\sqrt{x^2 - y^2}$
cosh	$e^z = \sinh z + \cosh z$
$\tanh^{-1}$	
$\times$	
$\div$	

Universal CORDIC

☺

Indirectly Computes :

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

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

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

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

$$\log_b w = K \times \ln w \qquad \sqrt{w} = \sqrt{(w+1/4)^2 - (w-1/4)^2}$$

$$w^t = e^{t \ln w}$$

Summary of Generalized CORDIC Algorithms

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

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

$z^{(i+1)} = z^{(i)} - d_i\, e^{(i)}$

$\mu \in \{-1, 0, 1\}$

$d_i \in \{-1, 1\}$

$K = 1.646\,760\,258\,121 \ldots$

$1/K = .607\,252\,935\,009 \ldots$

$K' = .828\,159\,360\,960\,2 \ldots$

$1/K' = 1.207\,497\,067\,763 \ldots$

☺

Mode →	Rotation: $d_i = \text{sign}(z^{(i)})$, $z^{(i)} \to 0$	Vectoring: $d_i = -\text{sign}(x^{(i)} y^{(i)})$, $y^{(i)} \to 0$
$\mu = 1$ 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 → $K \sqrt{x^2 + y^2}$ y → CORDIC → 0 z → CORDIC → $z + \tan^{-1}(y/x)$ For $\tan^{-1}$, set $x = 1$, $z = 0$ $\cos^{-1} w = \tan^{-1}[\sqrt{1 - w^2} / w]$ $\sin^{-1} w = \tan^{-1}[w / \sqrt{1 - w^2}]$
$\mu = 0$ 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 $z = 0$
$\mu = -1$ Hyperbolic $e^{(i)} = \tanh^{-1} 2^{-i}$	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 → $K' \sqrt{x^2 - y^2}$ y → CORDIC → 0 z → CORDIC → $z + \tanh^{-1}(y/x)$ For $\tanh^{-1}$, set $x = 1$, $z = 0$ $\ln w = 2 \tanh^{-1} \lvert (w - 1)/(w + 1) \rvert$ $\sqrt{w} = \sqrt{(w + 1/4)^2 - (w - 1/4)^2}$ $\cosh^{-1} w = \ln(w + \sqrt{1 - w^2})$ $\sinh^{-1} w = \ln(w + \sqrt{1 + w^2})$
Note →	In executing the iterations for $\mu = -1$, steps 4, 13, 40, 121, . . . , j , $3j + 1$, . . . must be repeated. These repetitions are incorporated in the constant K' below.	

Use of Approximating Functions

Convert the problem of evaluating the function *f* to that of function *g* approximating *f*, perhaps with a few pre- and postprocessing operations

Approximating polynomials need only additions and multiplications

Polynomial approximations can be derived from various schemes

The Taylor-series expansion of $f(x)$ about $x = a$ is

$f(x) = \sum_{j=0 \text{ to } \infty} f^{(j)}(a)\, (x - a)^j / j!$

The error due to omitting terms of degree > *m* is:

$f^{(m+1)}(a + \mu(x - a))\, (x - a)^{m+1} / (m + 1)! \qquad 0 < \mu < 1$

Setting $a = 0$ yields the Maclaurin-series expansion

$f(x) = \sum_{j=0 \text{ to } \infty} f^{(j)}(0)\, x^j / j!$

and its corresponding error bound:

$f^{(m+1)}(\mu x)\, x^{m+1} / (m + 1)! \qquad 0 < \mu < 1$

Source: Parhami

Some Polynomial Approximations

Func	Polynomial approximation	Conditions
$1/x$	$1 + y + y^2 + y^3 + \cdots + y^i + \cdots$	$0 < x < 2$, $y = 1 - x$
e^x	$1 + x/1! + x^2/2! + x^3/3! + \cdots + x^i/i! + \cdots$	
$\ln x$	$-y - y^2/2 - y^3/3 - y^4/4 - \cdots - y^i/i - \cdots$	$0 < x \le 2$, $y = 1 - x$
$\ln x$	$2[z + z^3/3 + z^5/5 + \cdots + z^{2i+1}/(2i+1) + \cdots]$	$x > 0$, $z = \frac{x-1}{x+1}$
$\sin x$	$x - x^3/3! + x^5/5! - x^7/7! + \cdots + (-1)^i x^{2i+1}/(2i+1)! + \cdots$	
$\cos x$	$1 - x^2/2! + x^4/4! - x^6/6! + \cdots + (-1)^i x^{2i}/(2i)! + \cdots$	
$\tan^{-1} x$	$x - x^3/3 + x^5/5 - x^7/7 + \cdots + (-1)^i x^{2i+1}/(2i+1) + \cdots$	$-1 < x < 1$
$\sinh x$	$x + x^3/3! + x^5/5! + x^7/7! + \cdots + x^{2i+1}/(2i+1)! + \cdots$	
$\cosh x$	$1 + x^2/2! + x^4/4! + x^6/6! + \cdots + x^{2i}/(2i)! + \cdots$	
$\tanh^{-1} x$	$x + x^3/3 + x^5/5 + x^7/7 + \cdots + x^{2i+1}/(2i+1) + \cdots$	$-1 < x < 1$

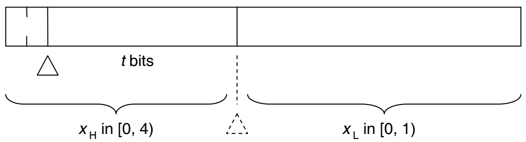
Function Evaluation via Divide-and-Conquer

Let x in [0, 4) be the $(l+2)$-bit significand of a floating-point number or its shifted version. Divide x into two chunks x_H and x_L:

$x = x_H + 2^{-t} x_L$

$0 \le x_H < 4$ $t+2$ bits

$0 \le x_L < 1$ $l-t$ bits



The Taylor-series expansion of $f(x)$ about $x = x_H$ is

$$f(x) = \sum_{j=0 \text{ to } \infty} f^{(j)}(x_H)\,(2^{-t} x_L)^j / j!$$

A linear approximation is obtained by taking only the first two terms

$$f(x) \cong f(x_H) + 2^{-t} x_L\, f'(x_H)$$

If t is not too large, f and/or f' (and other derivatives of f, if needed) can be evaluated via table lookup

Source: Parhami

Approximation by the Ratio of Two Polynomials

Example, yielding good results for many elementary functions

$$f(x) \cong \frac{a^{(5)}x^5 + a^{(4)}x^4 + a^{(3)}x^3 + a^{(2)}x^2 + a^{(1)}x + a^{(0)}}{b^{(5)}x^5 + b^{(4)}x^4 + b^{(3)}x^3 + b^{(2)}x^2 + b^{(1)}x + b^{(0)}}$$

Using Horner's method, such a "rational approximation" needs 10 multiplications, 10 additions, and 1 division

Source: Parhami

What is a Digital Biquad Filter?

- Transfer function:

$$H(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}}$$

- This can be rearranged as a difference equation:-

$$y_n = a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2} - b_1 y_{n-1} - b_2 y_{n-2}$$

- This can be generalised to an inner-product calculation:

$$y = [a_1 a_2 \ldots\ldots] \begin{bmatrix} x_1 \\ .. \\ x_N \end{bmatrix} \sum_{k=1}^{N} A_k x_k$$

Distributed Arithmetic (1)

- Let us express x_k in its 2's complement binary form:

$$x_k = -x_{k0} + x_{k1} 2^{-1} + x_{k2} 2^{-2} + \ldots\ldots + x_{k(B-1)} 2^{-(B-1)}$$
$$= -x_{k0} + \sum_{i=1}^{B-1} x_{ki} 2^{-i}$$

- Then:

$$y = \sum_{k=1}^{N} A_k \left[-x_{k0} + \sum_{i=1}^{B-1} x_{ki} 2^{-i} \right] = -\sum_{k=1}^{N} x_{k0} A_k + \sum_{k=1}^{N} \sum_{i=1}^{B-1} x_{ki} A_k 2^{-i}$$

Distributed Arithmetic (2)

- Let us expand this to:

MSB of x_N

$$\begin{aligned} y = & -\left[x_{10}A_1 + x_{20}A_2 + x_{30}A_3 + \dots\dots\dots + x_{N0}A_N\right] \\ & + \left[x_{11}A_1 + x_{21}A_2 + x_{31}A_3 + \dots\dots\dots + x_{N1}A_N\right]2^{-1} \\ & + \left[x_{12}A_1 + x_{22}A_2 + x_{32}A_3 + \dots\dots\dots + x_{N2}A_N\right]2^{-2} \\ & \bullet \\ & + \left[x_{1(B-2)}A_1 + x_{2(B-2)}A_2 + x_{3(B-2)}A_3 + \dots\dots\dots + x_{N(B-2)}A_N\right]2^{-(B-2)} \\ & + \left[x_{1(B-1)}A_1 + x_{2(B-1)}A_2 + x_{3(B-1)}A_3 + \dots\dots\dots + x_{N(B-1)}A_N\right]2^{-(B-1)} \end{aligned}$$

LSB of x1

LSB of xN

Use ROM as table lookup

- We can avoid any multiplication by table lookup:
  - Use (x_{1i}, x_{2i}, x_{3i}, ... x_{Ni}) as address to a ROM
  - Store pre-calculated partial product for each line in ROM:

$$\Phi(x_{1i}, x_{2i}, x_{3i} \dots, x_{Ni}) = A_1x_{1i} + A_2x_{2i} + A_3x_{3i} + \dots\dots\dots.. + A_Nx_{Ni}$$

- We can calculate y three operations: ROM lookup, shift, add/subtract:

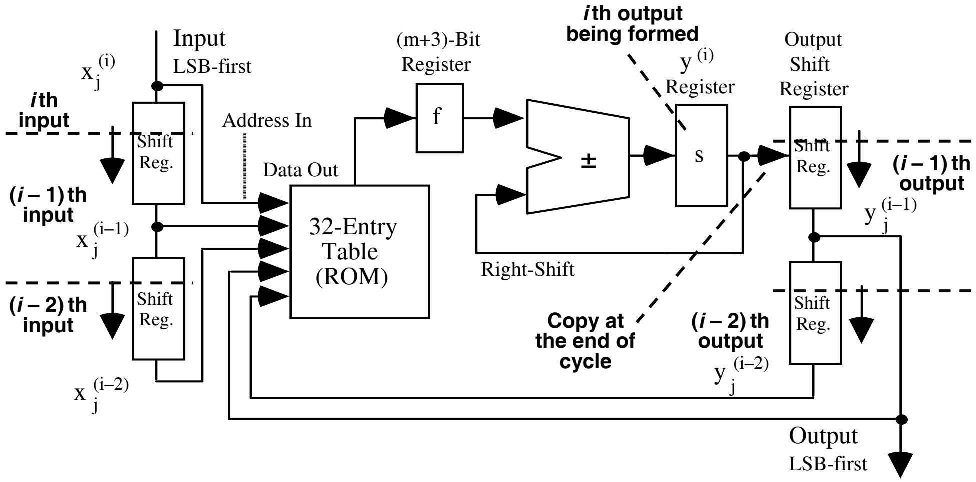
ROM lookup

$$y = -\Phi(x_{10}, x_{20}, x_{30} \dots, x_{N0}) + \sum_{i=1}^{B-1} 2^{-i}\Phi(x_{1i}, x_{2i}, x_{3i} \dots, x_{Ni})$$

Add/Sub

Shifting

Bit-Serial Implementation



Source: Parhami

References on CORDIC

- Volder J.E., "The CORDIC trigonometric comuting technique", IRE Trans. Electronic Computing, vol EC-8, 1959.
- Walther J.S., "A unifed algorithm for elementary functions", Spring Joint Computer Conference, 1971.
- Andraka R., "A survey of cordic algorithms for fpga based computers", Proc. Of 6th Int. symp. On FPGA, 1998.
- Par
- Schelin C.W., "Calculator Function Approximation", Am. Math. Monthly, vol.90, 1983.
- Lindlbauer N., "Application of FPGA's to Musical Gesture Communication and Processing", MS thesis, Berkeley, 1999.
- B. Parhami, "Computer Arithmetic", Chapter 22, OUP.

References on Distributed Arithmetic

- Peled and B. Liu, “A New Hardware Realization of Digital Filters”, *IEEE Trans. on Acoust., Speech, Signal Processing*, vol. ASSP-22, pp. 456-462, Dec. 1974.
- S. A. White, ``Applications of Distributed Arithmetic to Digital Signal Processing'', *IEEE ASSP Magazine*, Vol. 6(3), pp. 4-19, July 1989.
- “The Role of Distributed Arithmetic in FPGA-based Signal Processing”, *http://www.xilinx.com/appnotes/theory1.pdf*
- “Transposed form FIR Filter”, Xilinx App. Notes 219