

Low Complexity Arithmetic Implementation for DSP Radio Receivers

QIANG GAO

University of Strathclyde

qiang.gao@eee.strath.ac.uk



Summary

- Motivation
- CORDIC Fundamentals
- Parallel, “Pipelined Parallel”, and Serial CORDIC
- Barrel shifter based serial CORDIC design
- Conclusion
- Further work

Motivation

- Low power and complexity arithmetic unit plays an important role in Specknet Tx/Rx design in following areas:
 1. Oscillator design
 2. Constellation rotation
 3. Localization application

Introduction

- Various communication techniques require trigonometry, square root etc arithmetic operations.
- Implementing the arithmetic unit on digital circuit becomes an important research area
- In this presentation the low complexity CORDIC algorithm is introduced; this is a “shift and add” algorithm that allows calculation of many various trigonometric functions, such as: arctan, sine, cosine and other functions including divide and logarithmic functions.

Coordinate Rotation Digital Computer

- The Coordinate Rotation Digital Computer (CORDIC) algorithm is a simple technique using mainly shifts, additions and lookup tables (LUTs) – Ideal for hardware implementation
- Used to compute many different operations including trigonometric functions
- Based on Givens rotation
- Iterative technique

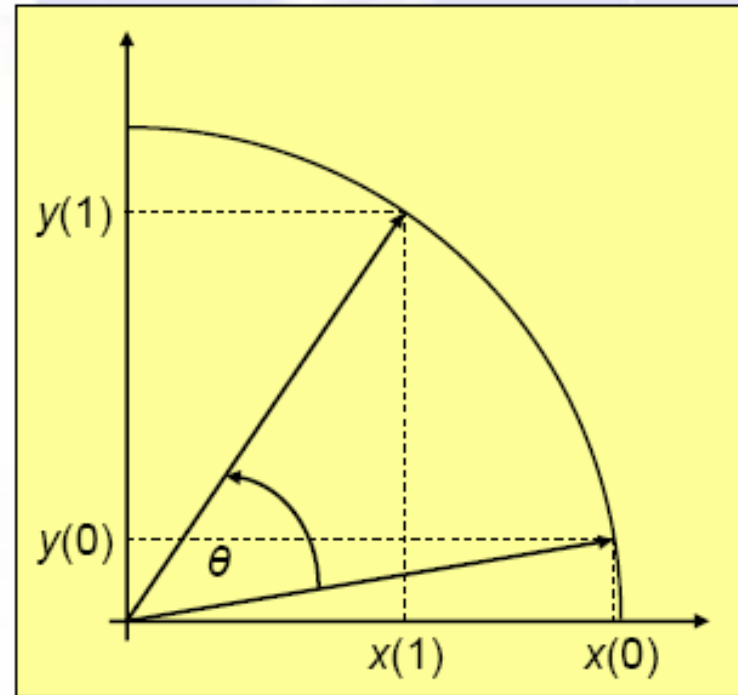
Givens Rotation

- CORDIC algorithm derived from Givens rotation:

$$\begin{aligned}x(1) &= x(0)\cos\theta - y(0)\sin\theta \\y(1) &= x(0)\sin\theta + y(0)\cos\theta\end{aligned}$$

Which can be written as:

$$\begin{aligned}x(1) &= \cos\theta(x(0) - y(0)\tan\theta) \\y(1) &= \cos\theta(y(0) + x(0)\tan\theta)\end{aligned}$$



- The pseudo-rotation is introduced by dropping the cosine term

Pseudo-Rotations

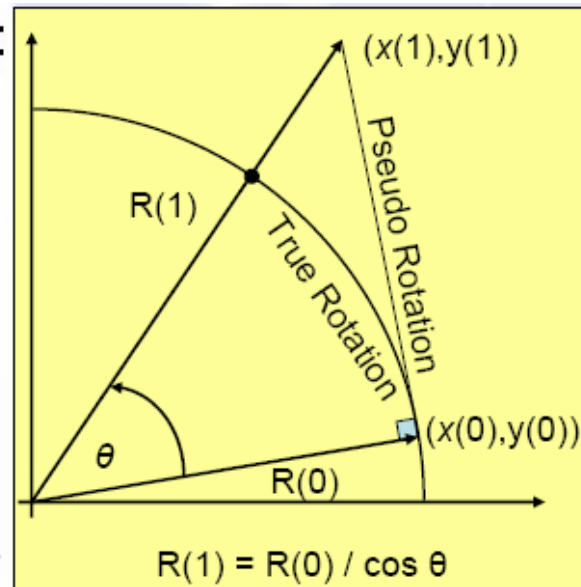
- The ***pseudo-rotation*** is introduced by performing a rotation: the angle of rotation is correct but the x and y values are scaled by $\cos\theta$ (i.e. both become larger than before as $1/\cos\theta > 1$).
- Note that it can make the computation of plane rotations more amenable to simple operations.

Dropping the $\cos\theta$ term gives:

$$\begin{aligned}x(1) &= x(0) - y(0)\tan\theta \\y(1) &= y(0) + x(0)\tan\theta\end{aligned}$$

Consequence is **pseudo rotation** rather than true rotation

Note scaling of original vector



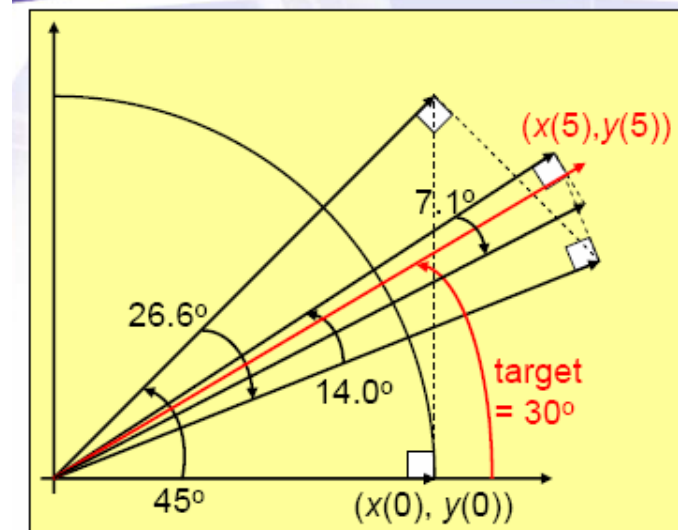
The CORDIC Method

- The key to the CORDIC method is to only (pseudo-) rotate by angles of θ_i where $\tan \theta_i = 2^{-i}$. Therefore in the equation:

$$\begin{aligned}x(i+1) &= x(i) \pm (2^{-i} y(i)) \\y(i+1) &= y(i) \pm (2^{-i} x(i))\end{aligned}$$

- The table below shows the rotation angles that can be used for each iteration (i) of the CORDIC algorithm:

Iteration (i)	$\tan \theta_i = 2^{-i}$	θ_i
0	1	45
1	0.5	26.6
2	0.25	14.0
3	0.125	7.1
4	0.0625	3.6



Shift-Add algorithm

- Original Givens rotation now reduced to iterative shift-add algorithm

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

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

$$z(i+1) = z(i) - d_i \theta_i$$

$$d_i = \pm 1$$

- Operations required are ideal for hardware implementation
- Thus each iteration requires:
 - 2 shifts
 - 1 lookup table (values)
 - 3 additions

Scaling Factor

- When simplifying the algorithm to allow pseudo-rotations the cosine term was omitted.
- Applying this to iterative scheme, the scaling factor K_n can be written as:

$$K_n = \prod_{i=0}^{n-1} \frac{1}{\cos \theta_i} = \prod_{i=0}^{n-1} \sqrt{1 + \tan^2 \theta_i} = \prod_{i=0}^{n-1} \sqrt{1 + 2^{-2i}}$$

- As the iteration number n is known, K_n can be precomputed
- Scaling factor removed by multiplying by $1/K_n$
- Therefore require additional multiplier in some cases

Rotation Mode

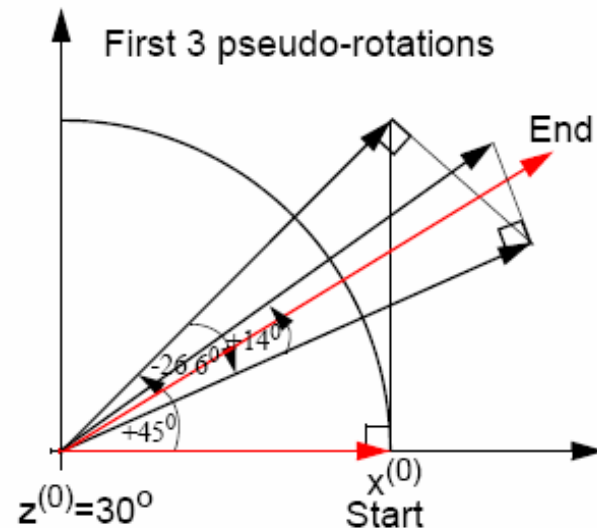
- The CORDIC method is operated in one of two modes;
- The mode of operation dictates the condition for the control operator di ;
- In Rotation Mode choose: $di = \text{sign}(z^{(i)})$, $z^{(i)} \rightarrow 0$;

- After n iterations we have:

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

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

$$z^{(n)} = 0$$



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

Cosine, Sine Calculation

- Finally, the magnitude of the rotated vector = $1/K_n * K_n$, so the value of y equals to the sine value of rotated angle.
- In the same way, x 's value = cosine

Vectoring Mode and Arctan Calculation

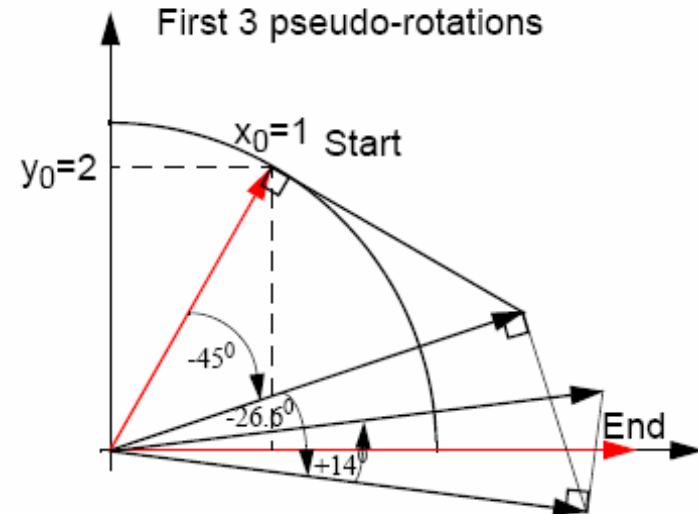
- In Vectoring Mode choose: $di = -\text{sign}(x^{(i)} y^{(i)})$, $y^{(i)} \rightarrow 0$

- After n iterations we have:

$$x^{(n)} = K_n \left(\sqrt{(x^{(0)})^2 + (y^{(0)})^2} \right)$$

$$y^{(n)} = 0$$

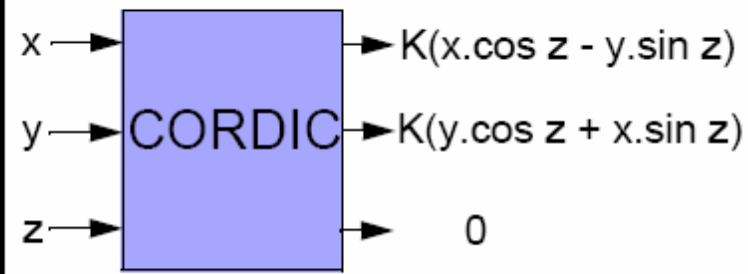
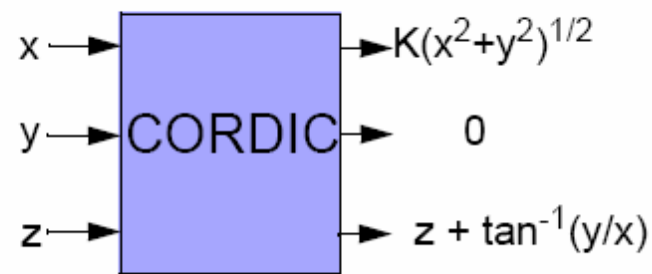
$$z^{(n)} = z^{(0)} + \tan^{-1} \left(\frac{y^{(0)}}{x^{(0)}} \right)$$



- Can compute $\tan^{-1} y^{(0)}$ by setting $x^{(0)} = 1$ and $z^{(0)} = 0$
- Final value of X after scaling is the magnitude of the vector
- Final accumulated angle is the arctan of the original vector

CORDIC output

- The output from the CORDIC algorithm operating in Rotation & Vectoring modes is:

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

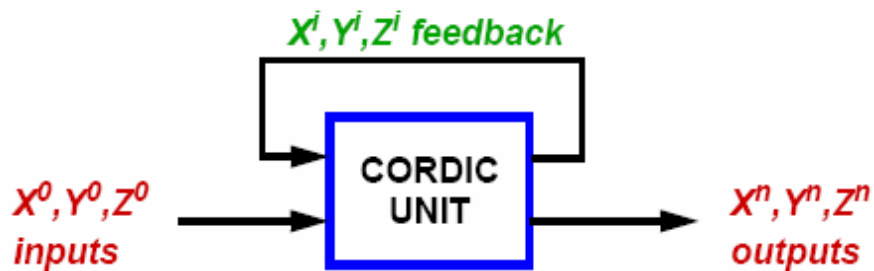
- The 'K' value is the scaling factor, which can be removed by multiplying the result by $1/K$.

Hardware Implementation

- The ideal CORDIC architecture depends on speed vs area tradeoffs in the intended application.
- Alternative implementation options will be presented, and the cost and performance trade-offs highlighted.
 1. **Rolled** - all CORDIC iterations performed using a single cell.
 2. **Unrolled** - an array of cells, one for each iteration.
 3. **Unrolled, pipelined** - retimed to improve performance.

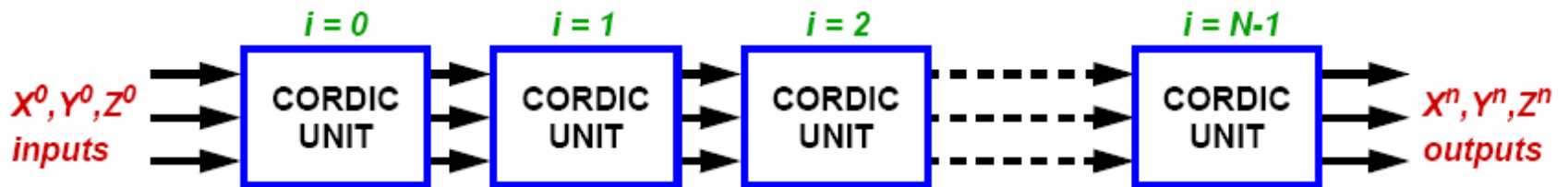
Implementation Architectures

1) Rolled

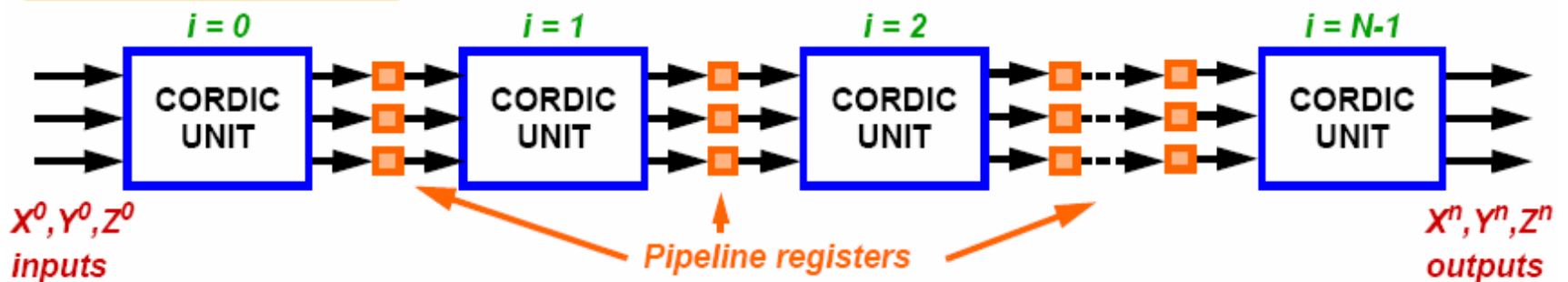


Note that the CORDIC unit in the rolled architecture is different compared to the unrolled designs: it must keep track of the iteration number at each clock cycle and synthesise the correct shift. As the CORDIC unit is shared N times, it runs at $1/N$ of the rate.

2) Unrolled



3) Unrolled and Pipelined

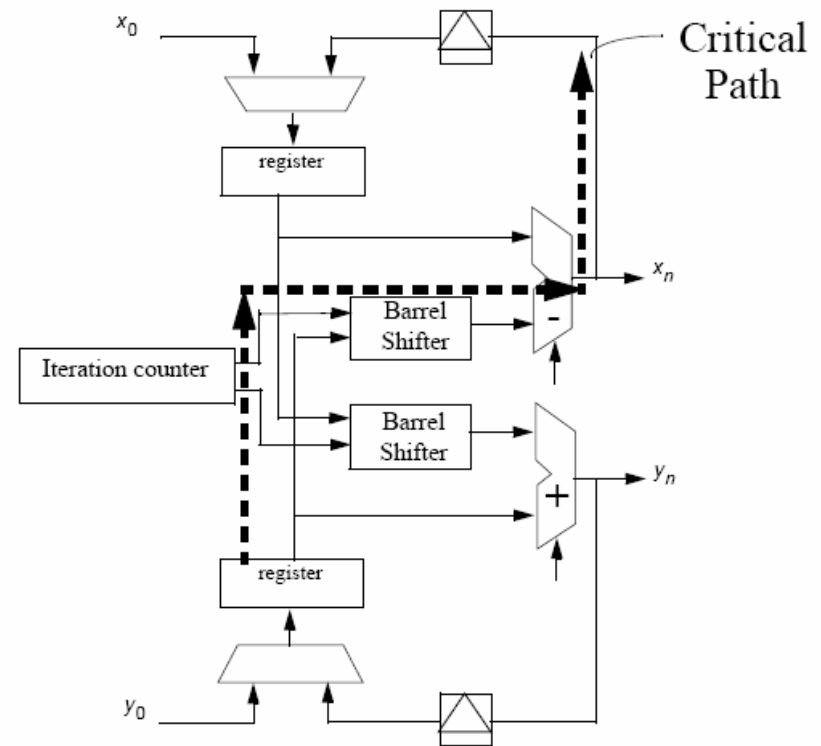


Single CORDIC Cell

- Often a dedicated rolled architecture is used for each iteration. It can lower the hardware cost of the CORDIC processor by time-sharing.

- The main cost is the variable length shift register required.

- The barrel shifter is core part of the critical path.



- Pipelined Barrel Shifter will cause much higher speed

Comparison of Hardware Cost

- Here FPGA is used as the prototyping platform to implement three types of CORDIC architecture (in vector mode).
- The FPGA resource cost of above three types of CORDIC is listed as follows (under the iteration number = 18)

	Parallel	Pipelined Parallel	Serial
Cost FPGA Resource (Slices)	368	718	223

- So serial CORDIC architecture is the cheapest way to execute arithmetic operations

Power Consumption Consideration

- Each CORDIC iteration cell is a Processing Element (PE)
 - The more iteration cells, the higher power consumption[1]
 - So single PE based serial CORDIC should have the lowest power consumption
1. A. Chandrakasan, S. Sheng, R. W. Brodersen, "Low-power CMOS design," IEEE Journal of Solid-State Circuits, vol 27, no.4, pp 472-484, April 1992

Conclusion

- Low complexity arithmetic unit plays an important role in communications/specknet system
- CORDIC algorithm has been presented
- The results of Parallel, versus "Pipelined Parallel", versus Serial CORDIC is listed.
- The serial CORDIC implementation using a barrel shifter has been described
- The pipelined Barrel Shifter based serial CORDIC suits to the low cost and power communication application in Specknet

Further Work

- Low cost and complexity CORDIC based square root IP design
- Single PE based CORDIC cell to execute different arithmetic operations together

For example:

