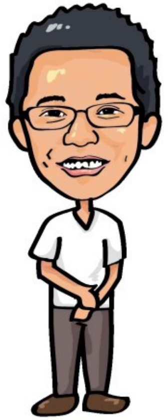
YunTech
National Yunlin University of Science & Technology

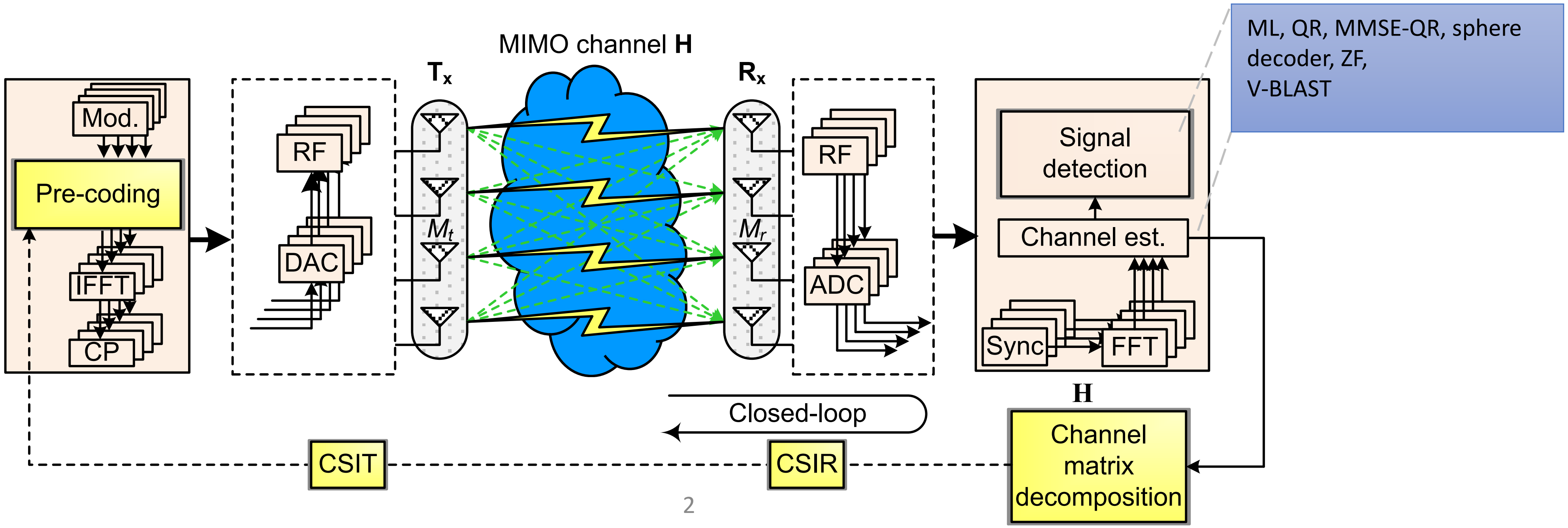
Signal detector on MIMO-OFDM system

Wei-Da Chen



MIMO signal detections(1)

- Signal detection for MIMO-OFDM systems
  - Signal detection is one of the most computing-intensive module
  - Roughly O(N^3) complexity



2

MIMO signal detections(2)

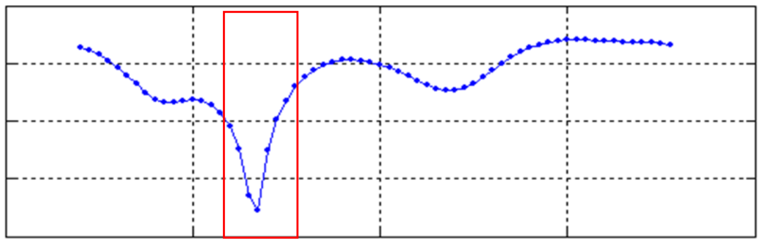
- Communication baseband involves a lot of DSP computations and the complexity increases with
  - Number of antenna
  - Bandwidth and data rate
- MIMO signal detection and precoding are two of the most computationally intensive module
- Efficient DSP computing algorithms are crucial to the system performance
- Effective hardware designs facilitate low complexity and real time system implementations

Zero-forcing in MIMO-OFDM systems

- Zero-forcing algorithm
  - Inverse of channel H
    - H is matrix of n by n
    - x =Hs+v

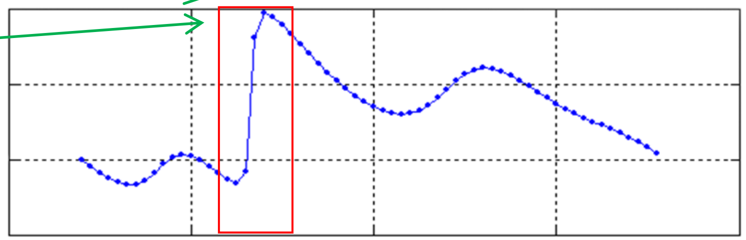
      $H^{-1}x=s+\boxed{H^{-1}v}$
  - Pseudo-inverse of channel H
    - H is matrix of n by m
    - x =Hs+v

      $((H^*H)^{-1}H^*)x=s+\boxed{(H^*H)^{-1}H^*v}$
  - Advantages and drawbacks
    - Noise enhancement and poor efficacy.
    - Low hardware resources.



Channel response

Noise enhancement



Channel response is inverted

QR-blast for MIMO-OFDM systems

- QR decomposition
  - Step 1:Doing QR decomposition of H: H=QR

$$y = Hx + n$$

$$Q^H y = Q^H QRx + \boxed{Q^H n}$$

Noise power doesn't increase because Q is orthogonal matrix.

$$y' = Rx + n$$

$$\begin{bmatrix} y_1' \\ y_2' \\ \vdots \\ y_N' \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & \cdots & H_{1N} \\ 0 & H_{21} & \cdots & H_{2N} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & H_{NN} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} + \begin{bmatrix} n_1 \\ n_2 \\ \vdots \\ n_N \end{bmatrix}$$

  - Step 2:Backward substitution

$$\hat{x}_i = Quantise\left\{ \frac{1}{H_{i,i}} \left(y_i' - \sum_{k=i+1}^{N} H_{i,k} \hat{x}_k \right) \right\}$$

YunTech 國立雲林科技大學

QR decomposition

- Complex QR factorization can be applied to spherical decoder ,zero forcing and QR-blast in the future.
  - Spherical decoder approaches maximum likelihood performance.
- Complex QR factorization methods:
  - Given rotation
    - This method is more easily parallelized than Householder and Gram-Schmidt transformations.
    - To use CORDIC technique solves problems of given rotation .
  - Gram Schmidt
    - It can use complex divider, multiplication and square root.
  - Householder
    - It can use complex divider, multiplication and square root.
    - This method has greater numerical stability than the Gram-Schmidt method.

Given rotation(1)

- Real given rotation
  - Channel has been estimated before compute complex QR factorization.
    -

$$complex\_H = \begin{bmatrix} a+ja & c+jc \\ b+jb & d+jd \end{bmatrix}$$

  - Complex_H matrix changes from complex form to real.
    - Complex_H will be magnify 2n-by-2n from n-by-n.

$$\mathrm{Real\_H} = \begin{bmatrix} R(complex\_H) & -I(complex\_H) \\ I(complex\_H) & R(complex\_H) \end{bmatrix}$$

$$= \begin{bmatrix} a & c & -ja & -jc \\ b & d & -jb & -jd \\ ja & jc & a & c \\ jb & jd & b & d \end{bmatrix}$$

Given rotation(2)

- QR decomposition can be computed with a series of Given rotations.
  - Step1.

$$\text{Real\_H}^1\text{=Rot}_1\text{Real\_H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(\theta_1) & \sin(\theta_1) \\ 0 & 0 & -\sin(\theta_1) & \cos(\theta_1) \end{bmatrix} \begin{bmatrix} a & c & -ja & -jc \\ b & d & -jb & -jd \\ ja & jc & a & c \\ jb & jd & b & d \end{bmatrix} = \begin{bmatrix} a & c & -ja & -jc \\ b & d & -jb & -jd \\ ja^1 & jc^1 & a^1 & c^1 \\ 0 & jd^1 & b^1 & d^1 \end{bmatrix}$$

  - Step2

$$\text{Real\_H}^2\text{=Rot}_2\text{Real\_H}^1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta_1) & \sin(\theta_1) & 0 \\ 0 & -\sin(\theta_1) & \cos(\theta_1) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a & c & -ja & -jc \\ b & d & -jb & -jd \\ ja^1 & jc^1 & a^1 & c^1 \\ 0 & jd^1 & b^1 & d^1 \end{bmatrix} = \begin{bmatrix} a & c & -ja & -jc \\ b^1 & d^1 & -jb^1 & -jd^1 \\ 0 & jc^2 & a^2 & c^2 \\ 0 & jd^1 & b^1 & d^1 \end{bmatrix}$$

  - Step3~step6.

$$\text{Real\_H}^6\text{=Rot}_6\text{Rot}_5\text{Rot}_4\text{Rot}_3 \begin{bmatrix} a & c & -ja & -jc \\ b^1 & d^1 & -jb^1 & -jd^1 \\ 0 & jc^2 & a^2 & c^2 \\ 0 & jd^1 & b^1 & d^1 \end{bmatrix} = \begin{bmatrix} a^1 & c^1 & -ja^1 & -jc^1 \\ 0 & d^3 & -jb^3 & -jd^3 \\ 0 & 0 & a^5 & c^5 \\ 0 & 0 & 0 & d^3 \end{bmatrix}$$

✓CORDIC numbers:
- Vectoring mode:6
- Rotation mode:14

Complex-valued Given rotation(1)

- Classical complex given rotation algorithm.
  - Matrix H changes from complex form to pole.

$$Pole\_H = \begin{bmatrix} a + ja & c + jc \\ b + jb & d + jd \end{bmatrix} = \begin{bmatrix} Ae^{j\theta_a} & Ce^{j\theta_c} \\ Be^{j\theta_b} & De^{j\theta_d} \end{bmatrix}$$

  - Rotation matrix will be build when Pole_H is finished.

$$Rotation\_matrix = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 \, e^{j(\theta_a - \theta_b)} \\ -\sin\theta_1 \, e^{-j(\theta_a - \theta_b)} & \cos\theta_1 \end{bmatrix} \qquad \theta_1 = \tan^{-1}\left(\frac{B}{A}\right)$$

  - Rotation_matrix multiplied by Pole_H is upper triangle.

$$\begin{bmatrix} \cos\theta_1 & \sin\theta_1 \, e^{j(\theta_a - \theta_b)} \\ -\sin\theta_1 \, e^{-j(\theta_a - \theta_b)} & \cos\theta_1 \end{bmatrix} \begin{bmatrix} Ae^{j\theta_a} & Ce^{j\theta_c} \\ Be^{j\theta_b} & De^{j\theta d} \end{bmatrix} = \begin{bmatrix} \mathrm{X}e^{j\theta_x} & Ye^{j\theta_y} \\ 0 & Ze^{j\theta_z} \end{bmatrix}$$

Complex-valued Given rotation(2)

- Modified classical complex given rotation algorithm.

$$Rotation\_matrix = \begin{bmatrix} e^{-j\theta_a} & 0 \\ 0 & e^{-j\theta_b} \end{bmatrix} \begin{bmatrix} \cos\theta_1 & \sin\theta_1 e^{j(\theta_a - \theta_b)} \\ -\sin\theta_1 e^{-j(\theta_a - \theta_b)} & \cos\theta_1 \end{bmatrix} = \begin{bmatrix} \cos\theta_1 e^{-j\theta_a} & \sin\theta_1 e^{-j\theta_b} \\ -\sin\theta_1 e^{-j\theta_a} & \cos\theta_1 e^{-j\theta_b} \end{bmatrix}$$

  - Matrix will lead to the appearance of real elements on the matrix diagonal.

$$\begin{bmatrix} \cos\theta_1 e^{-j\theta_a} & \sin\theta_1 e^{-j\theta_b} \\ -\sin\theta_1 e^{-j\theta_a} & \cos\theta_1 e^{-j\theta_b} \end{bmatrix} \begin{bmatrix} Ae^{j\theta_a} & Ce^{j\theta_c} \\ Be^{j\theta_b} & De^{j\theta d} \end{bmatrix} = \begin{bmatrix} \boxed{\mathrm{X}} & Ye^{j\theta_y} \\ 0 & Ze^{j\theta_z} \end{bmatrix}$$

  - To produce a matrix with only real diagonal elements.

$$\begin{bmatrix} 1 & 0 \\ 0 & e^{-j\theta_z} \end{bmatrix} \begin{bmatrix} Ae^{j\theta_a} & Ce^{j\theta_c} \\ Be^{j\theta_b} & De^{j\theta d} \end{bmatrix} = \begin{bmatrix} \mathrm{X} & Ye^{j\theta_y} \\ 0 & \boxed{Z} \end{bmatrix}$$

Exercise - QR decomposition(1)

- Design a 4x4 real-valued QR decomposition
  - A given 4x4 matrix is decomposed into two items through a series of CORDIC operations. One is called an orthogonal matrix **Q**, the other is an upper triangular matrix **R**.
  - The matrix **Q** is formed by the accumulative multiplication of a series of rotation matrixes.
  - The matrix **R** is fashioned by multiplying a series of rotation matrixes.

$$\text{Real\_H}^1 = \text{Rot}_1 \text{Real\_H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(\theta_1) & \sin(\theta_1) \\ 0 & 0 & -\sin(\theta_1) & \cos(\theta_1) \end{bmatrix} \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \\ h_{41} & h_{42} & h_{43} & h_{44} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \\ 0 & h_{42} & h_{43} & h_{44} \end{bmatrix}$$

Exercise - QR decomposition(2)

$$\text{Real\_H}^2 = \text{Rot}_2\text{Rot}_1\text{Real\_H} =$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta_2) & \sin(\theta_2) & 0 \\ 0 & -\sin(\theta_2) & \cos(\theta_2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(\theta_1) & \sin(\theta_1) \\ 0 & 0 & -\sin(\theta_1) & \cos(\theta_1) \end{bmatrix} \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \\ h_{41} & h_{42} & h_{43} & h_{44} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ 0 & h_{32} & h_{33} & h_{34} \\ 0 & h_{42} & h_{43} & h_{44} \end{bmatrix}$$

$$\mathbf{R} = \text{Rot}_6\text{Rot}_5\text{Rot}_4\text{Rot}_3 \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ 0 & h_{32} & h_{33} & h_{34} \\ 0 & h_{42} & h_{43} & h_{44} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ 0 & h_{22} & h_{23} & h_{24} \\ 0 & 0 & h_{33} & h_{34} \\ 0 & 0 & 0 & h_{44} \end{bmatrix}$$

$$\mathbf{Q} = \mathbf{Rot}_6 \cdot \mathbf{Rot}_5 \cdot \mathbf{Rot}_4 \cdot \mathbf{Rot}_3 \cdot \mathbf{Rot}_2 \cdot \mathbf{Rot}_1$$

H=QR

Introduction of CORDIC(1)

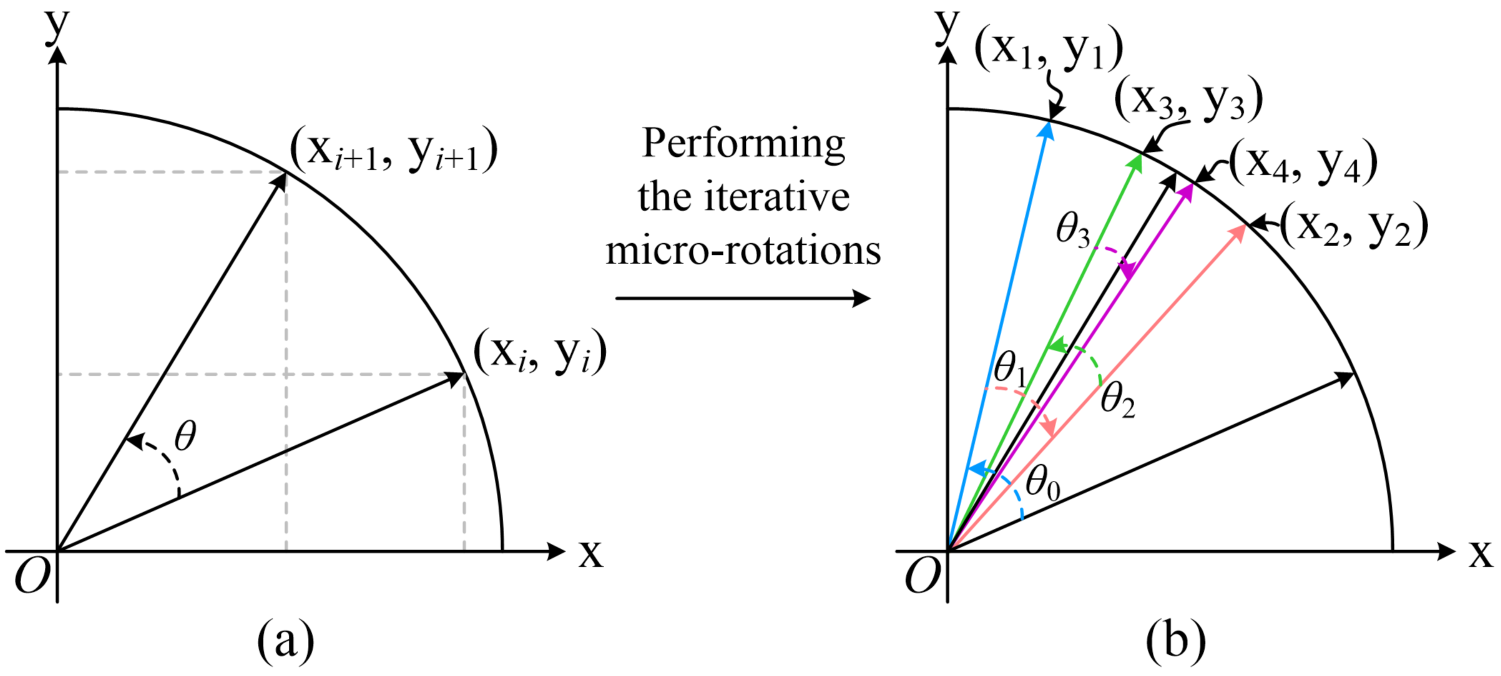
- All of operations, calculating values of cosine and sine and performing rotation matrices and square roots, can be implemented by low complexity CORDIC (COordinate Rotations Digital Computer) which has two types of computing modes.
  - The vectoring mode is to generate an included angle θ when a two-dimensional vector (x_i, y_i) is given.
  - The rotation mode is to rotate a two-dimensional vector (x_{i+1}, y_{i+1}) when an angle θ is given
- The original algorithm is expressed, as follows:

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} \cos\theta & \mp\sin\theta \\ \pm\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

the matrix is called the rotation matrix.

Introduction of CORDIC(2)

- A CORDIC module cannot rotate directly a vector from (x_i, y_i) to (x_{i+1}, y_{i+1}) but rather is gradually close to a desired position (x_{i+1}, y_{i+1}) by rotating micro-angles



2-dimensional plane of rotations

Introduction of CORDIC(3)

- Any rotation angle θ can be consisted of a series of micro-angles $\theta_1, \theta_2, .., \theta_n$.
  -

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \prod_{i=0}^{n-1} \begin{bmatrix} \cos\theta_i & \mp\sin\theta_i \\ \pm\sin\theta_i & \cos\theta_i \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \prod_{i=0}^{n-1} \cos\theta_i \cdot \prod_{i=0}^{n-1} \begin{bmatrix} 1 & \mp\tan\theta_i \\ \pm\tan\theta_i & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

- Trigonometric values for sine and cosine of the rotation angle are expressed by the simpler adder/subtraction and constant multiplier, instead of directly obtaining values of sine and cosine by using expensive hardware.

The K is called the scaling factor.

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \boxed{K} \cdot \prod_{i=0}^{n-1} \begin{bmatrix} 1 & \boxed{-\sigma_i} \cdot 2^{-i} \\ \sigma_i \cdot 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

The direction of rotation.
- counterclockwise rotation: +1
- clockwise rotation: -1

Introduction of CORDIC(4)

- The CORDIC algorithm can be summarized, as follows:

$$X_{i+1} = X_i - \sigma_i \cdot 2^{-i} \cdot Y_i$$

$$Y_{i+1} = Y_i + \sigma_i \cdot 2^{-i} \cdot X_i$$

$$Z_{i+1} = Z_i - \sigma_i \cdot \tan^{-1}(2^{-i})$$

$$K = \prod_{i=0}^{n-1} 1 \Big/ \sqrt{1 + 2^{-2i}}$$

- Z denotes the accumulated angle during all iterations, and the approximate relations between $\tan^{-1}(2^{-i})$.

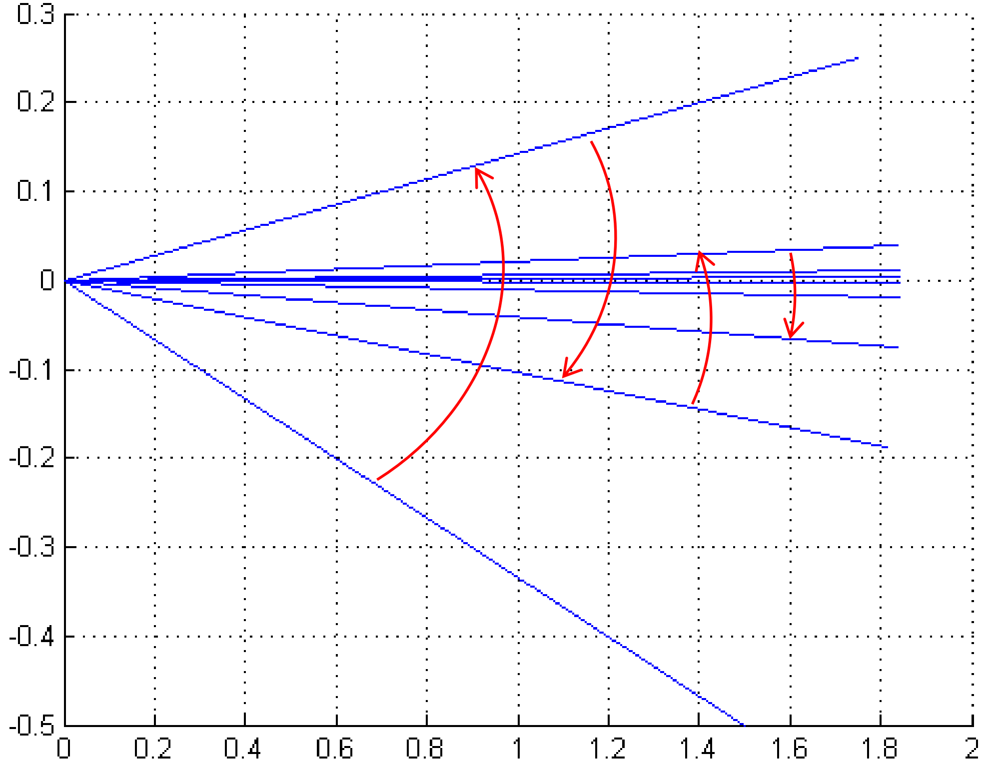
Approximate angle table

i	0	1	2	3	4	5	6	7
$\tan^{-1}(2^{-i}) = \theta$	45°	26.6°	14°	7.1°	3.6°	1.8°	0.9°	0.4°

CORDIC algorithm- vectoring mode

- CORDIC algorithm(1)
  - Vectoring mode
    - $$X_{i+1} = X_i - \sigma_i \cdot 2^{-i} \cdot Y_i$$
      $$Y_{i+1} = Y_i + \sigma_i \cdot 2^{-i} \cdot X_i$$
      $$\mathrm{Z}_{i+1} = \mathrm{Z}_i - \sigma_i \cdot \tan^{-1}(2^{-i})$$
    - Where

      $$\sigma_i = +1 \; if \; \mathrm{Y}_i < 0, -1 \; otherwise$$

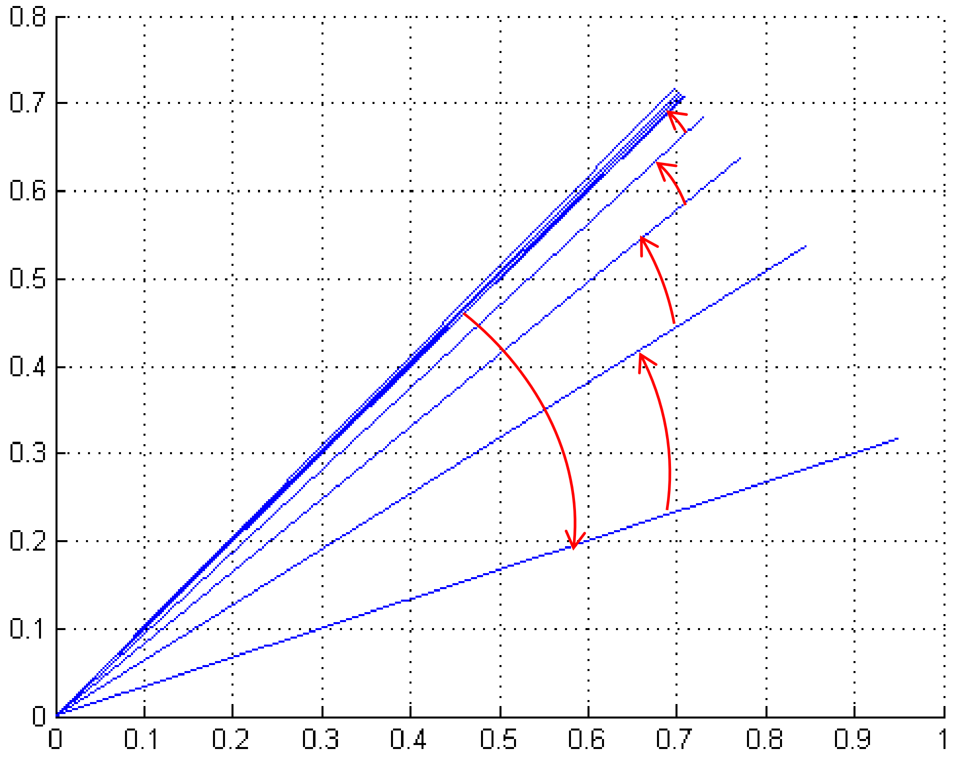


YunTech
國立雲林科技大學

CORDIC algorithm- rotation mode

- CORDIC algorithm(2)
  - Rotation mode
    - $$X_{i+1} = X_i - \sigma_i \cdot 2^{-i} \cdot Y_i$$
      $$Y_{i+1} = Y_i + \sigma_i \cdot 2^{-i} \cdot X_i$$
      $$\mathrm{Z}_{i+1} = \mathrm{Z}_i - \sigma_i \cdot \tan^{-1}(2^{-i})$$
    - Where

$$\sigma_i = +1 \; if \; \mathrm{Z}_i < 0, -1 \; otherwise$$



Examples for vectoring mode

```
function [ x_out,y_out,ang ] = vectoring_mode_cordic(x_value,...
                                    y_value,iteration )

ang = 0;
factor = 1;
for ii=1:iteration
    t = 2^-(ii-1);
    fac_tmp = (1/sqrt(1+(t^2)));
    factor = factor * fac_tmp;
end


for ii=1:iteration
     sign_value = -sign(y_value*x_value);
    t = 2^-(ii-1);
    x1 = x_value - sign_value*y_value*t;
    y1 = y_value + sign_value*x_value*t;
    ang = ang - sign_value*atand(t);
    x_value = x1;
    y_value = y1;
end

x_out = x_value*factor;
y_out = y_value*factor;
```

```
>> [ x_out,y_out,ang ] = vectoring_mode_cordic( 1,1,3)

ang =

    45


ang =

   71.5651


ang =

   57.5288
```

```
>> [ x_out,y_out,ang ] = vectoring_mode_cordic( 1,1,20)

x_out =

    1.4142


y_out =

   2.3160e-06


ang =

   44.9999
```

Examples for rotation mode

```
function [ x_out,y_out ] = rotating_mode_cordic_v1(...
                          x_value,y_value,iteration,ang)


factor = 1;
for ii=1:iteration
    t = 2^-(ii-1);
    fac_tmp = (1/sqrt(1+(t^2)));
    factor = factor * fac_tmp;
end

if ang~=0
    for ii=1:iteration
        t = 2^-(ii-1);
        sign_value = sign(ang);
        x1 = x_value - sign_value*y_value*t;
        y1 = y_value + sign_value*x_value*t;
        x_value = x1;
        y_value = y1;
        ang = ang - sign_value*atand(t);
    end
end

x_out = x_value*factor;
y_out = y_value*factor;
```

```
>> [ x_out,y_out] = rotating_mode_cordic_v1(1,0,30,90)

x_out =

   1.5266e-09


y_out =

    1.0000
```

```
>> [ x_out,y_out] = rotating_mode_cordic_v1(1,0,30,-45)

x_out =

    0.6073


y_out =

   -0.6073
```

Parallel CORDIC design

- CORDIC operates rotation and vectoring mode at the same time.
  - Combining vectoring with rotation mode.
    - $$Vec\_x_{i+1} = Vec\_x_i - Vec\_y_i \cdot \sigma_i \cdot 2^{-i}$$ $$Vec\_y_{i+1} = Vec\_y_i + Vec\_x_i \cdot \sigma_i \cdot 2^{-i}$$ $$Rot\_x_{i+1} = Rot\_x_i - Rot\_y_i \cdot \sigma_i \cdot 2^{-i}$$ $$Rot\_y_{i+1} = Rot\_y_i + Rot\_x_i \cdot \sigma_i \cdot 2^{-i}$$ $$z_{i+1} = z_i - \sigma_i \cdot \tan^{-1}(2^{-i})$$

Vectoring section

Rotation section

Saving LUTs.

- where

$$\sigma_i = +1\ if\ y_i < 0, -1\ otherwise$$

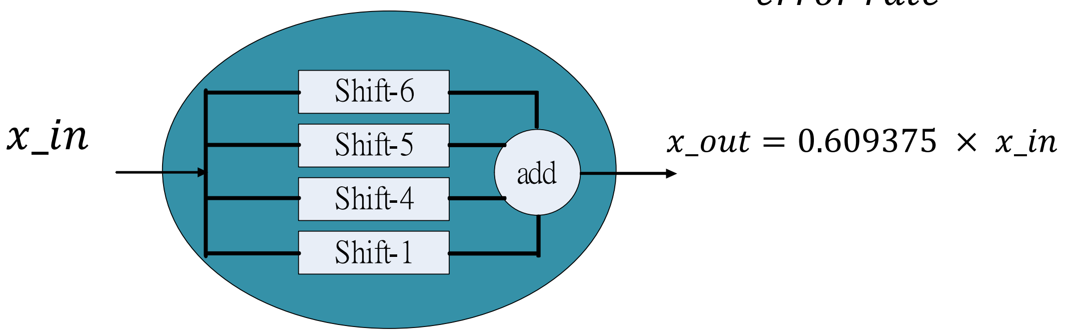
Scale factor of CORDIC design

- Scale factor algorithm
  - Output of CORIDC is divided by a scale factor k when rotation finished.

$$k = \prod_{n=0}^{n=7} \cos(\tan^{-1}(2^{-n})) = \frac{1}{\prod_{n=0}^{n=7}\sqrt{1+2^{-2n}}}$$
$$= 0.6072$$

  - Using shift-adder instead of divider or multiplication.

$$error\ rate = \frac{0.609375 - 0.6072}{0.6072} = 3.58\%$$



Exercise - QR decomposition(3)

- Content of homework
  - A 1000 different values of 4x4 matrix is provided.
  - Verify the algorithm of 4x4 real-valued QR decomposition by using CORDIC and decide the number of iterations and word length.
  - Implement the hardware of the decomposition by RTL code.
  - Offering area and frequency of the synthesized RTL code.
  - The implementation loss is defined as the following:

$$implementation\ loss = \left(\sum_{i=1}^{16000} \frac{R(i)_{ALG} - R(i)_{RTL}}{R(i)_{ALG}}\right) x100\%$$