Vitis HLS and CORDIC arithmetic

Joseph Cavallaro
Rice University
18 October 2022

Last Lectures

- Before Fall Break – Vitis HLS flows
- Help Session on Project 3

Today

- Begin Computer Arithmetic for CORDIC
  - ⇒ Sine and Cosine functions in hardware
  - ⇒ Vector or Givens rotations of a vector in hardware
  - ⇒ Useful for communication systems “up converters” and for matrix linear algebra functions

Vitis HLS Example on Canvas

- The Vectoradd_Vitis_HLS_MC in CAD_Tool_Examples
  - ⇒ vectoradd.cpp
  - ⇒ vectoradd.h
  - ⇒ vectoradd_test.cpp - Functions for Vitis HLS

  - ⇒ vectoradd_MC_HLS_default.slx - Model Composer file that can include the output of Vitis HLS. The Vitis HLS block needs to load the solution1 from your area instead of my default location.
  - ⇒ loadvector.m - Matlab script to run to provide data for the Model Composer file.

Computer Arithmetic

- Addition and Multiplication well studied.
  - ⇒ Fixed point and Floating point flavors
  - ⇒ Subtraction fits nicely with addition
- Division is problematic
  - ⇒ Often avoided through algorithm modification
- Square Root also problematic
  - ⇒ Often done in “software” or specialized “seed” instructions as in TI DSPs.
- Elementary Functions
  - ⇒ Not so elementary…..
  - ⇒ Sine, Cosine, Tangent, Hyperbolic, Logarithm…

CORDIC Arithmetic

- Similar to multiplication and division with shift and add or shift and subtract.
- Iterative method for “coordinate rotation” in a “digital computer.”
- Can help to find sine, cosine, inverse tangent, and vector rotation.
- Useful later on for “Givens rotations”

- Model Composer had multiple blocks, general and specialized. Currently, only CORDIC 6.0 appears in the library.

Readings on CORDIC Algorithm and Xilinx Versions

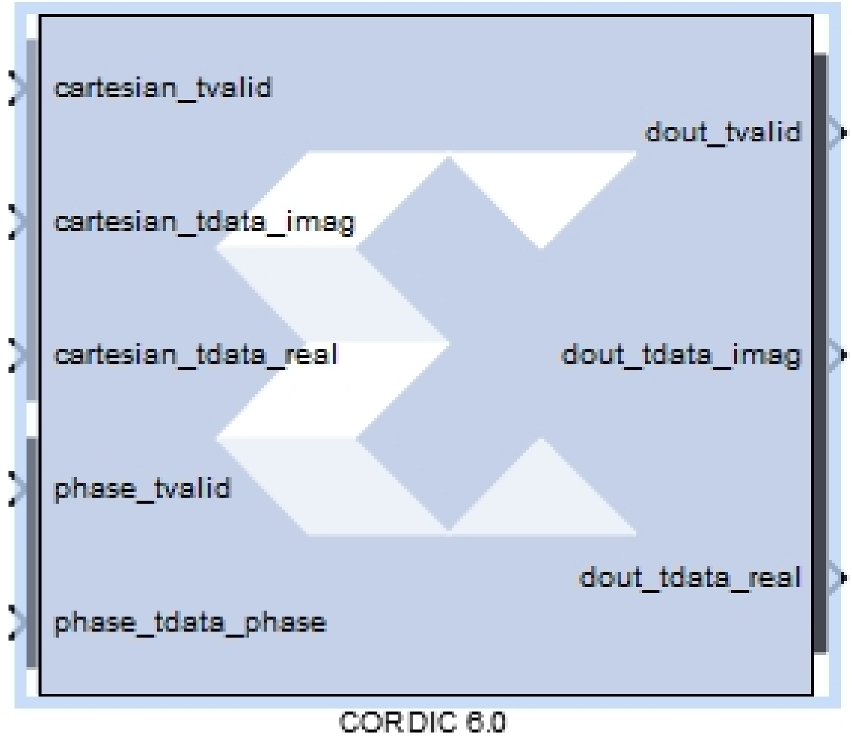
- Algorithm paper:
- W08_Walther_CORDIC.pdf

- Xilinx versions in CAD_Tools_Examples\CORDIC_Model_Composer:
- pg105-cordic.pdf

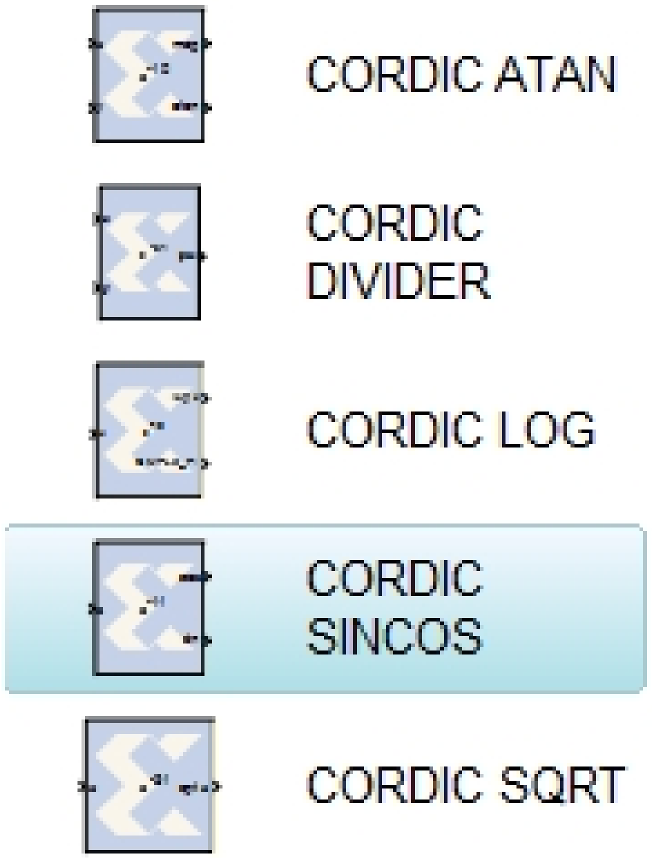
- CORDIC 6.0 is the current module in Model Composer

CORDIC Arithmetic in Model Composer

- Xilinx Math Library for Model Composer contains multi-purpose CORDIC 6 Module



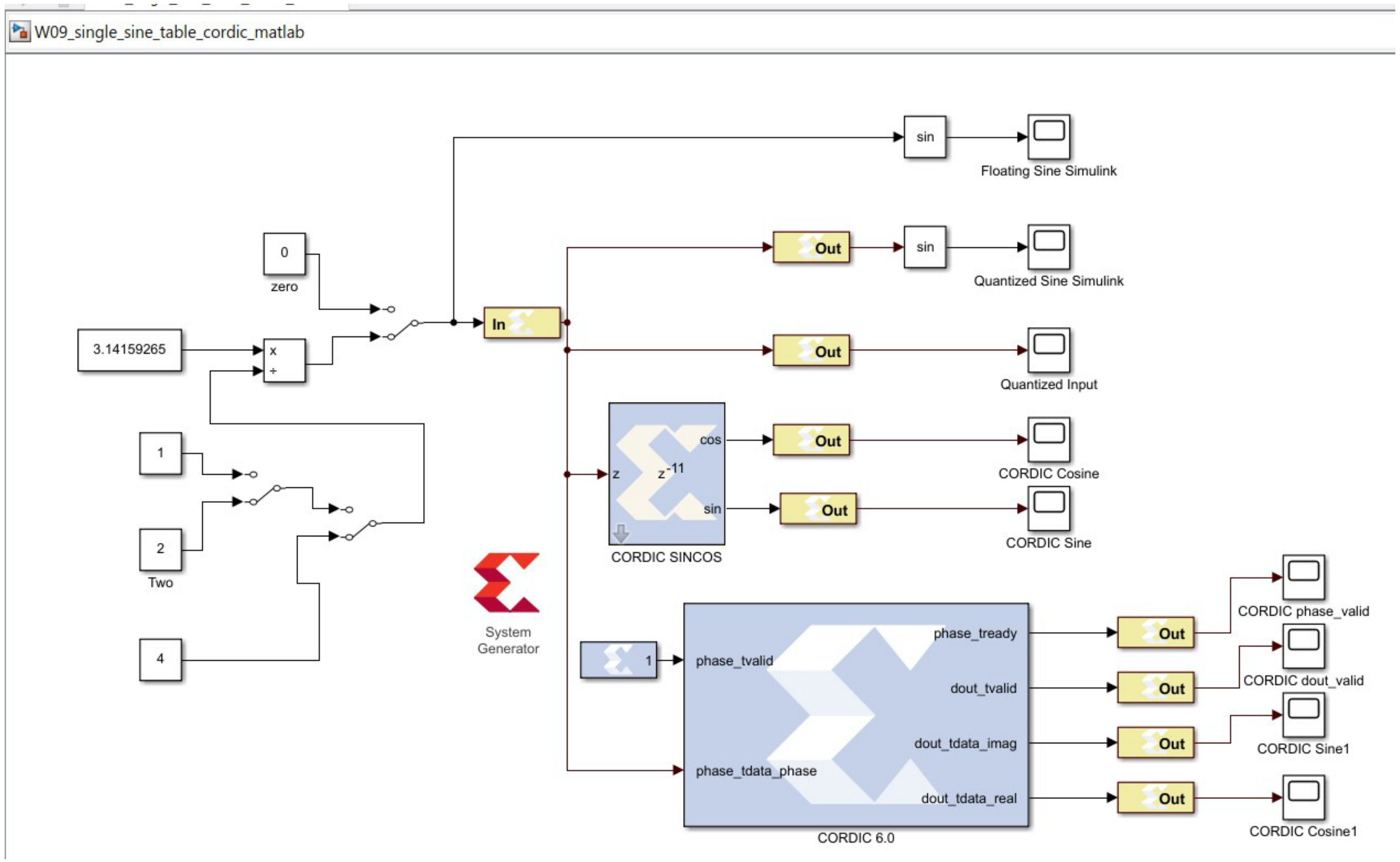
Previously Specialized CORDIC Functions in Model Composer – Focus on SINCOS



No Longer available in the Library

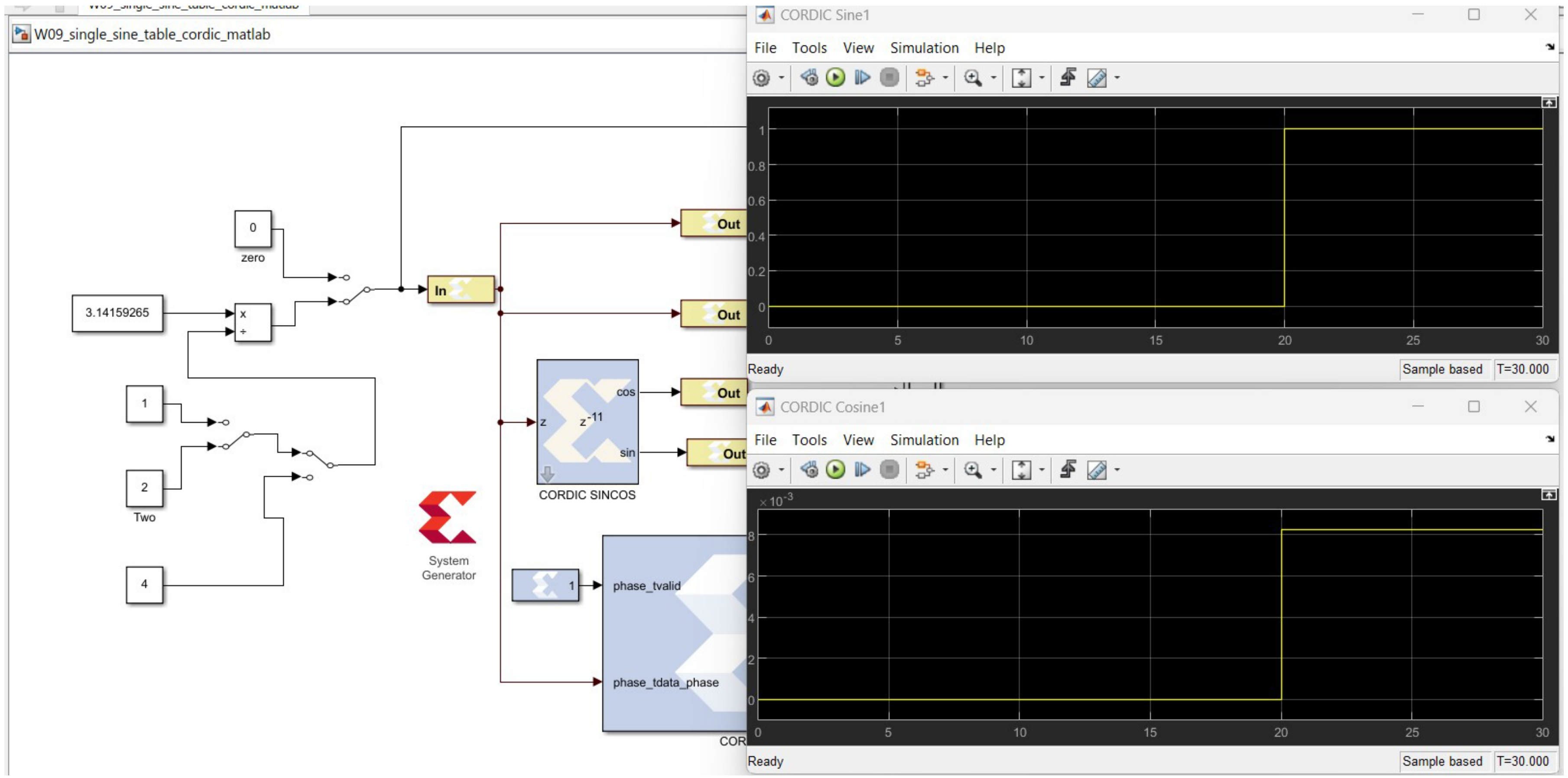
CORDIC – Test Module Demo – Canvas .slx file

- CAD_Example_Files/W09_single_sine_table_cordic_matlab.slx

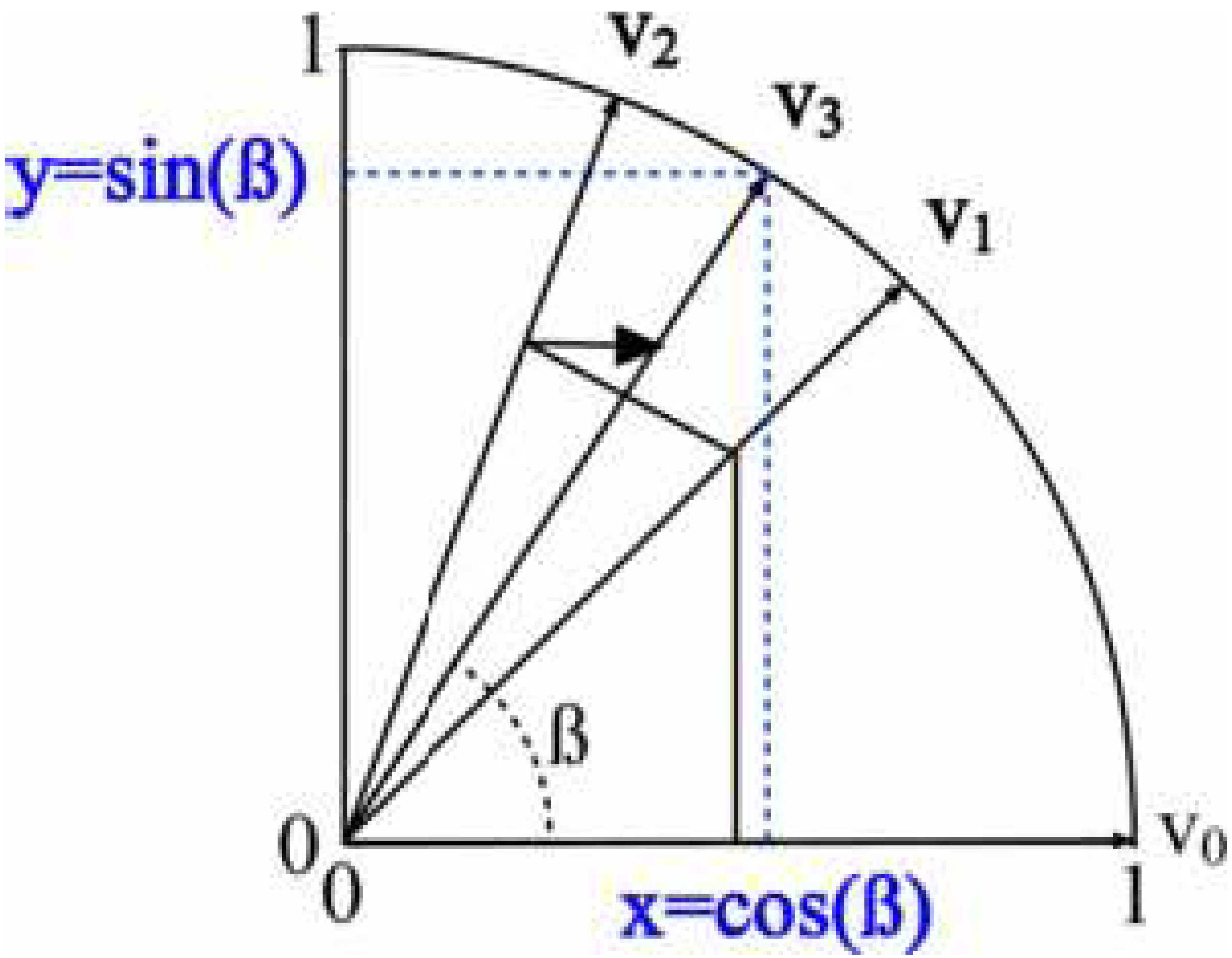


Pi/2 CORDIC Sine and Cosine

- 16 bit resolution yields about 5 decimal digits of accuracy
- About one bit per iteration
- Module takes 20 time steps in this example
- Sine approaches 1
- Cosine approaches 8x10^-3



CORDIC Arithmetic



CORDIC Equations from Plane Rotation

$$v_o = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$v_{i+1} = R_i v_i$$

$$R_i = \begin{pmatrix} \cos\gamma_i & -\sin\gamma_i \\ \sin\gamma_i & \cos\gamma_i \end{pmatrix}$$

$$v_{i+1} = R_i v_i = \cos\gamma_i \begin{pmatrix} 1 & -\sigma_i \tan\gamma_i \\ \sigma_i \tan\gamma_i & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

CORDIC Equation for Hardware

$$v_{i+1} = R_i v_i = \cos(\arctan(2^{-i}))\begin{pmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{pmatrix}\begin{pmatrix} x_i \\ y_i \end{pmatrix} = K_i \begin{pmatrix} x_i - \sigma_i 2^{-i} y_i \\ x_i \sigma_i 2^{-i} + y_i \end{pmatrix}$$

$$K_i = \cos(\arctan(2^{-i}))$$

$$K(n) = \prod_{i=0}^{n-1} K_i = \prod_{i=0}^{n-1} \cos(\arctan(2^{-i})) = \prod_{i=0}^{n-1} 1/\sqrt{1 + 2^{-2i}}$$

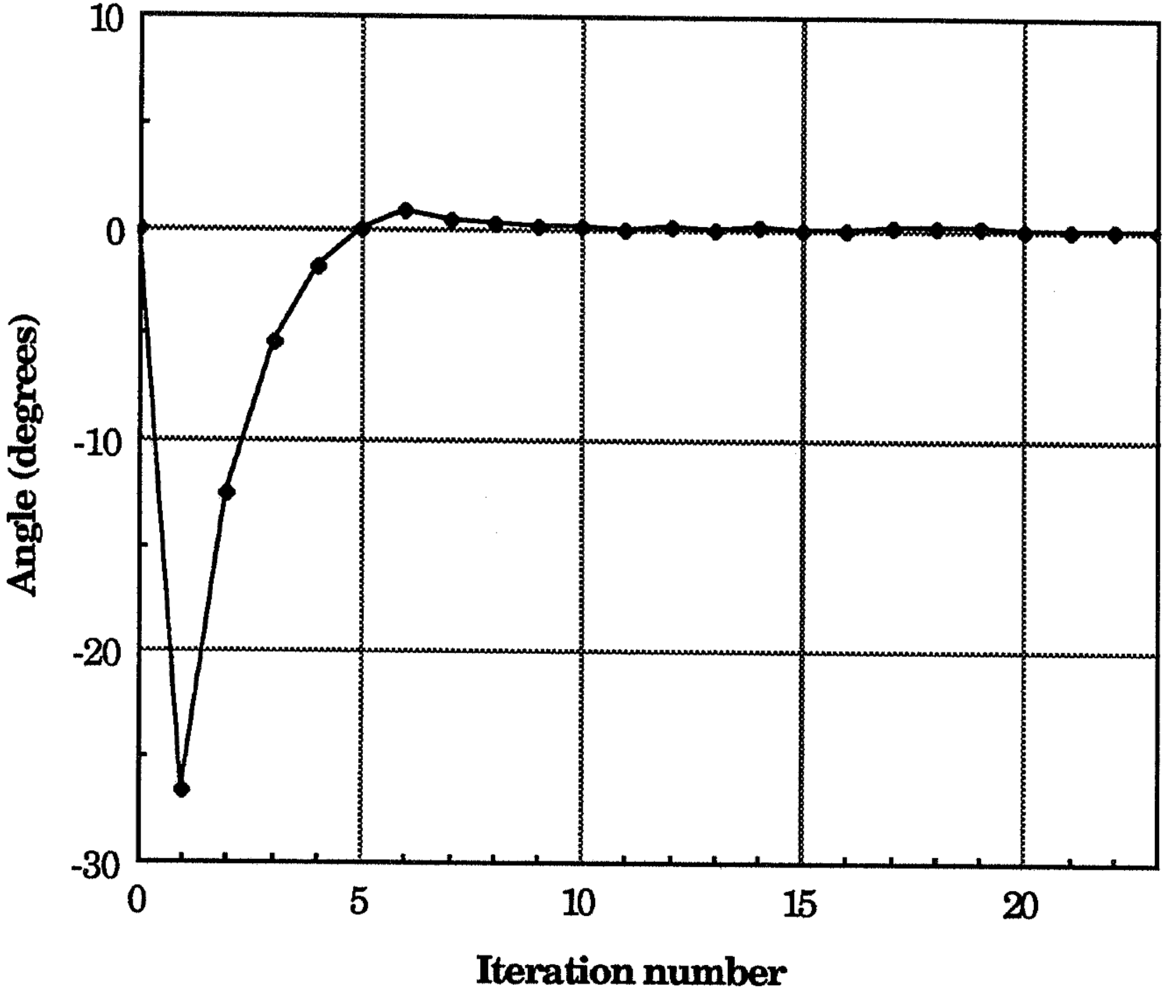
$$K = \lim_{n \to \infty} K(n) \approx 0.607252935$$

CORDIC Rotation Angle Table

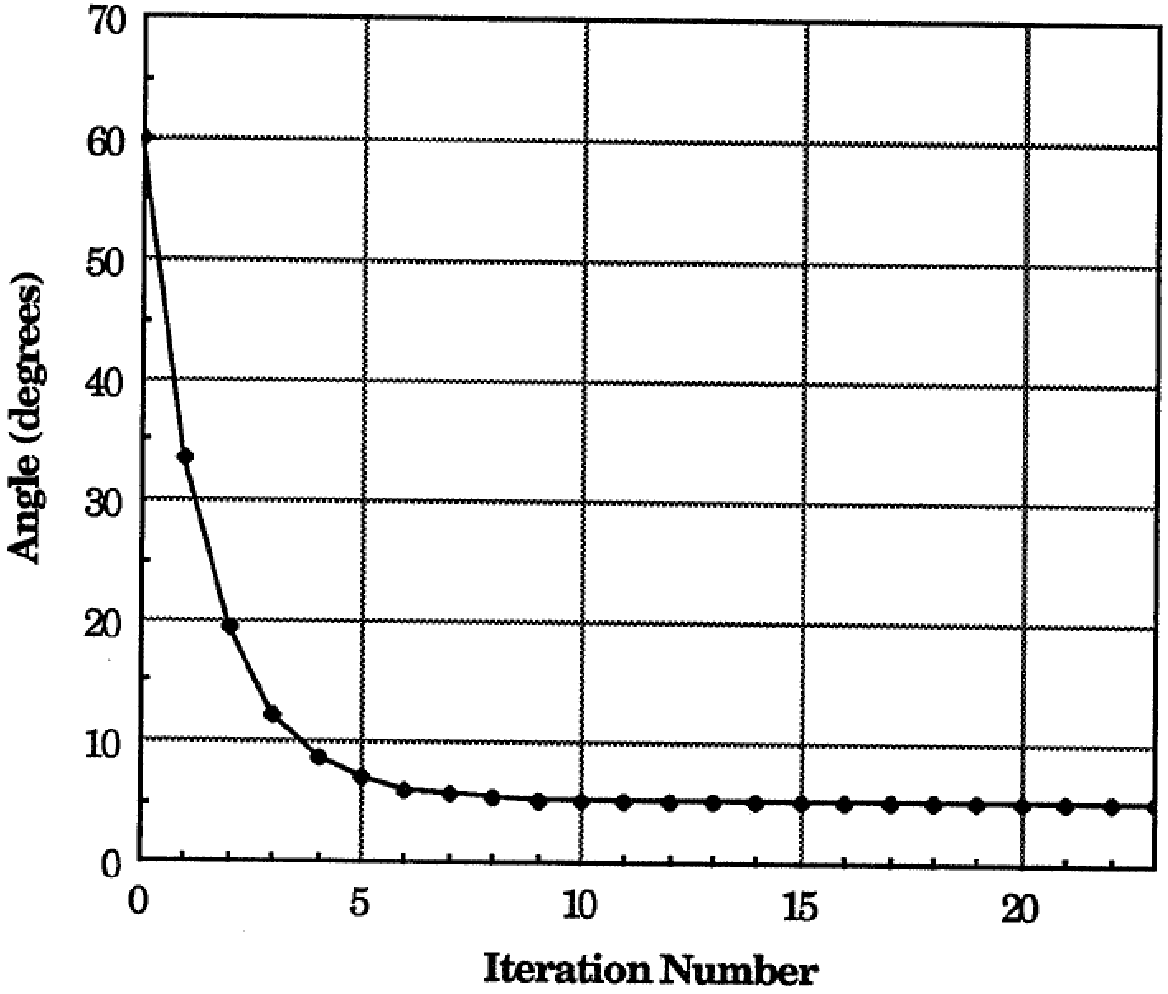
Rotation Angles for Circular Mode, $n = 24$

Iteration	Angle
0	44.9999999999
1	26.5650511770
2	14.0362434679
3	7.1250163489
4	3.5763343750
5	1.7899106082
6	0.8951737102
7	0.4476141709
8	0.2238105004
9	0.1119056771
10	0.0559528919
11	0.0279764526
12	0.0139882271
13	0.0069941137
14	0.0034970569
15	0.0017485284
16	0.0008742642
17	0.0004371321
18	0.0002185661
19	0.0001092830
20	0.0000546415
21	0.0000273208
22	0.0000136604
23	0.0000068302

Circular Mode Convergence for 45 Degrees



Circular Mode Non-convergence for 105 Degrees



CORDIC Results in Traditional Circular Mode

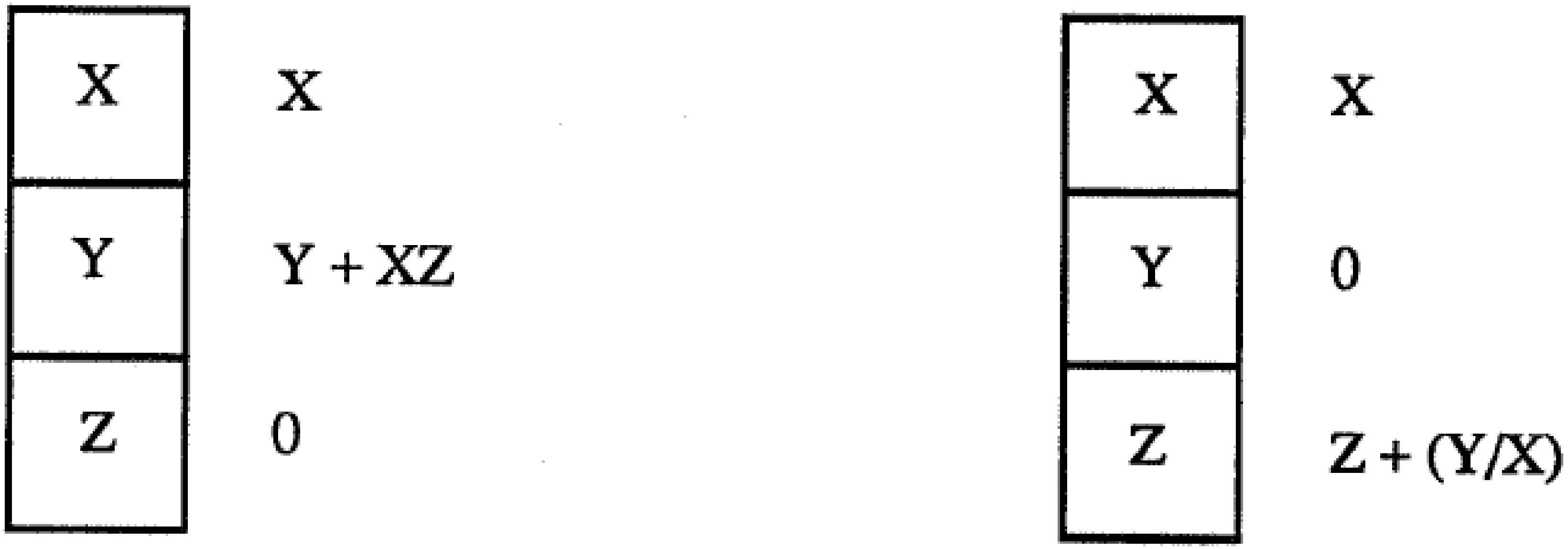
CIRCULAR MODE

X	$K(X\cos Z - Y\sin Z)$
Y	$K(Y\cos Z + X\sin Z)$
Z	0

X	$K(X^2 + Y^2)^{1/2}$
Y	0
Z	$Z + \arctan(Y/X)$

CORDIC Results in Linear Mode – Limited Range

LINEAR MODE



CORDIC Results in Hyperbolic Mode – based on Hyperbolic Tangent

HYPERBOLIC MODE

X	$K(X\cosh Z + Y\sinh Z)$
Y	$K(Y\cosh Z + X\sinh Z)$
Z	0

Z Reduction to 0

X	$K(X^2 - Y^2)^{1/2}$
Y	0
Z	$Z + \operatorname{arctanh}(Y/X)$

Y Reduction to 0

Y-Reduction to Find Inverse Tangent

- Scale factor cancels out in finding Inverse Tangent.

$$x_n = K_n(x_0 + y_0 \tan\theta),$$

$$y_n = K_n(y_0 - x_0 \tan\theta),$$

$$z_n = z_0 + \theta.$$

If, after n iterations, $y_n = 0$, and if $z_0 = 0$, then $\tan\theta = (y_0 / x_0)$ and

$$z_n = \tan^{-1}\left(\frac{y_0}{x_0}\right).$$

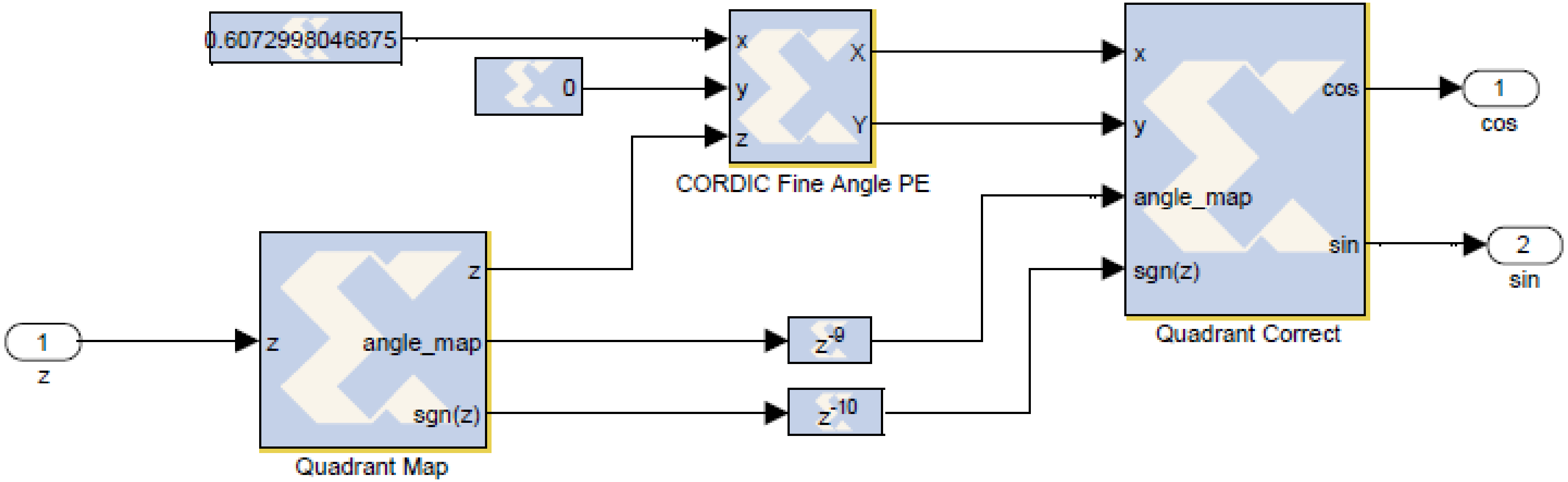
Y-Reduction C-code Declarations

```
yreduction (x, y, z, m, numiter)

double    *x;
double    *y;
double    *z;
double    m;          /* Mode: 1=circular; 0=linear; -1=hyperbolic */
int       numiter;    /* Number of iterations. */

{
extern    double angles[wordlength];
extern    double shifts[wordlength];
double    delta;
double    xnew;
double    ynew;
int       i;
```

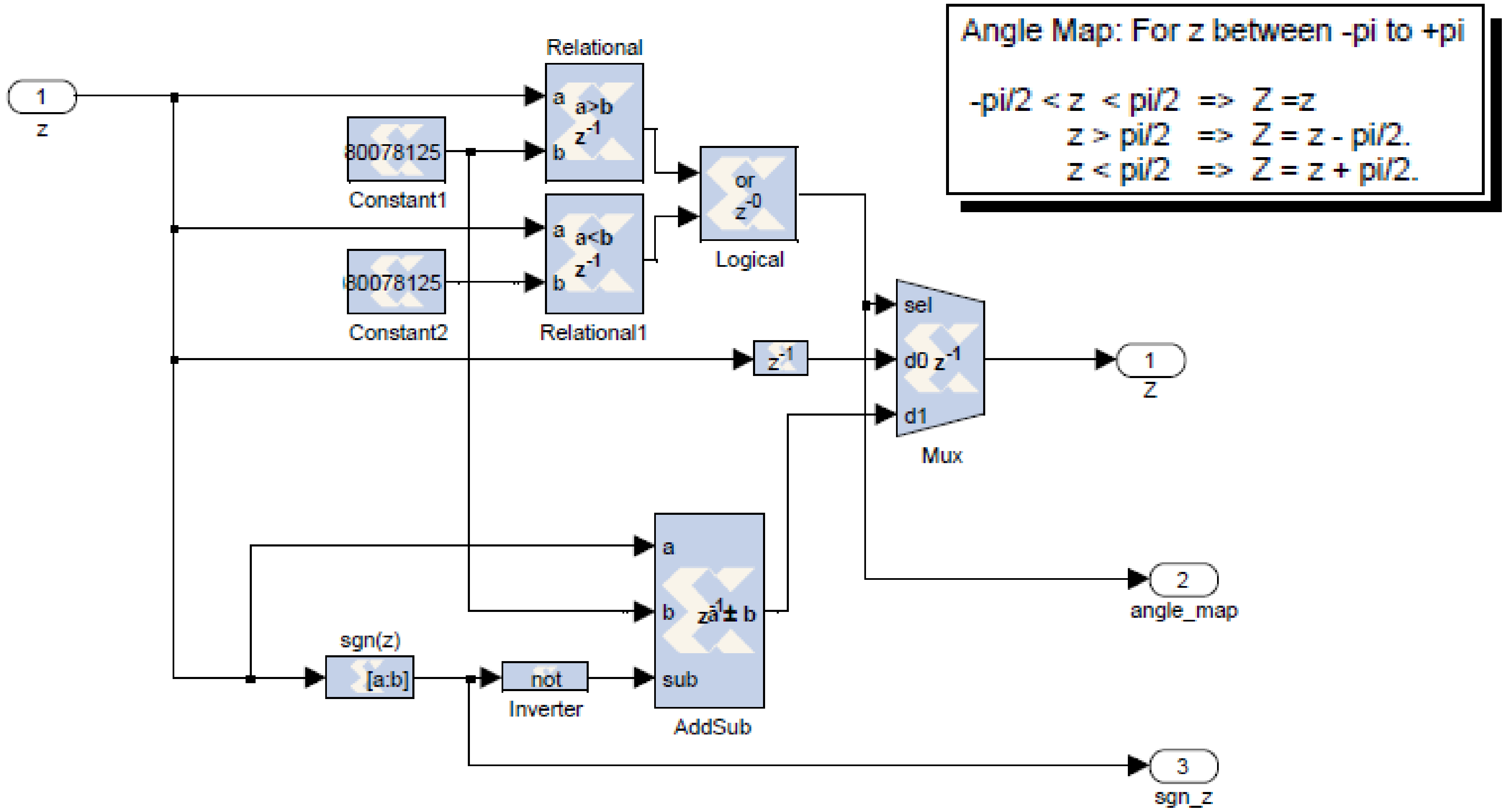
Y-Reduction C-Code Loop Body

```
if (((*x < 0)&&(*y < 0)) || ((*x < 0)&&(*y > 0)))
{
        *x = -*x;
        *y = -*y;
}
for ( i = 0; i < numiter; i++)
{
        if ( *y >= 0.0)
                delta = 1.0;
        else
                delta = -1.0;
        /* Calculate new x, y, and z.  */
        /* Basic CORDIC Rotations.  */
        xnew = *x + (m * (delta * (shifts[i] * *y)));
        ynew = *y - (delta * (shifts[i] * *x));
        *x = xnew;
        *y = ynew;
        *z = *z + (delta * angles[i]);
}
}
```

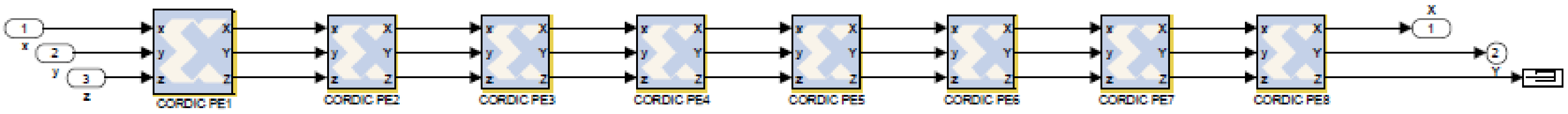
Model Composer "Old" CORDIC Block - Top



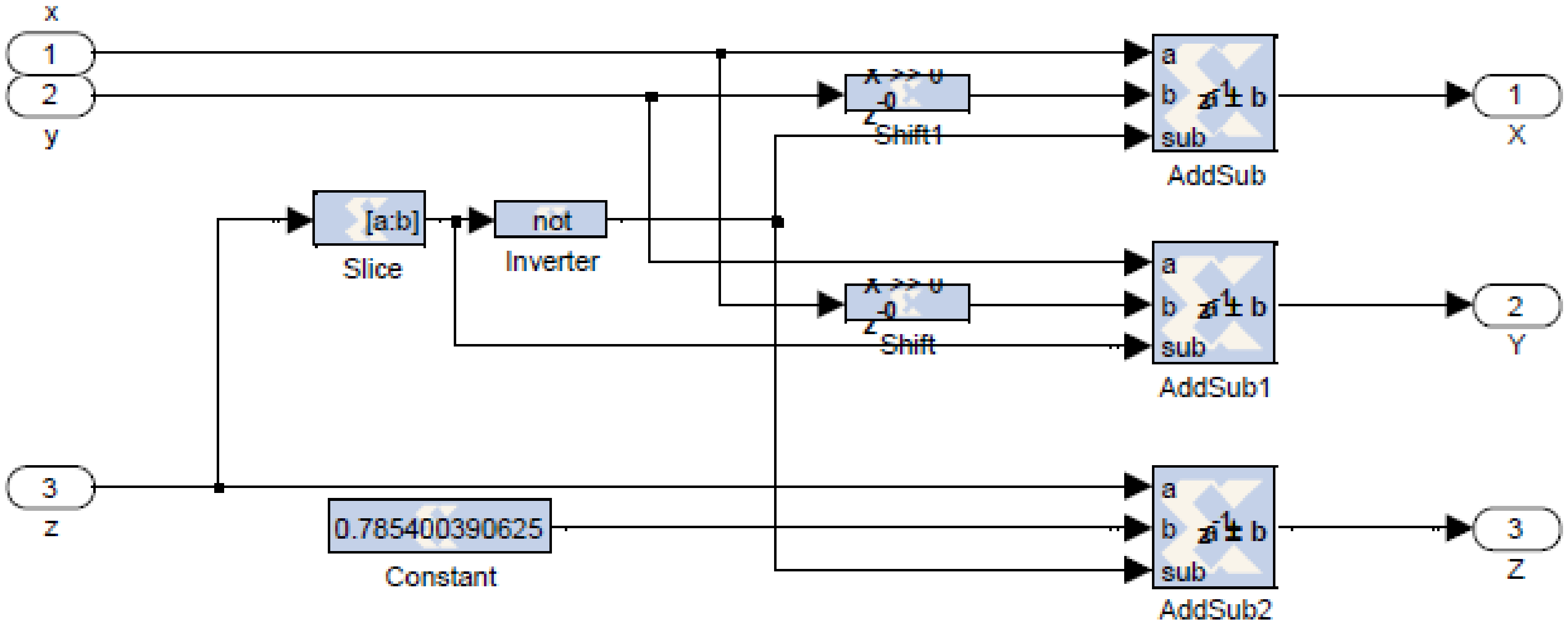
CORDIC Pre-Processing for Sign and Angle Quadrant



CORDIC Micro-rotation Core for 8 Iterations

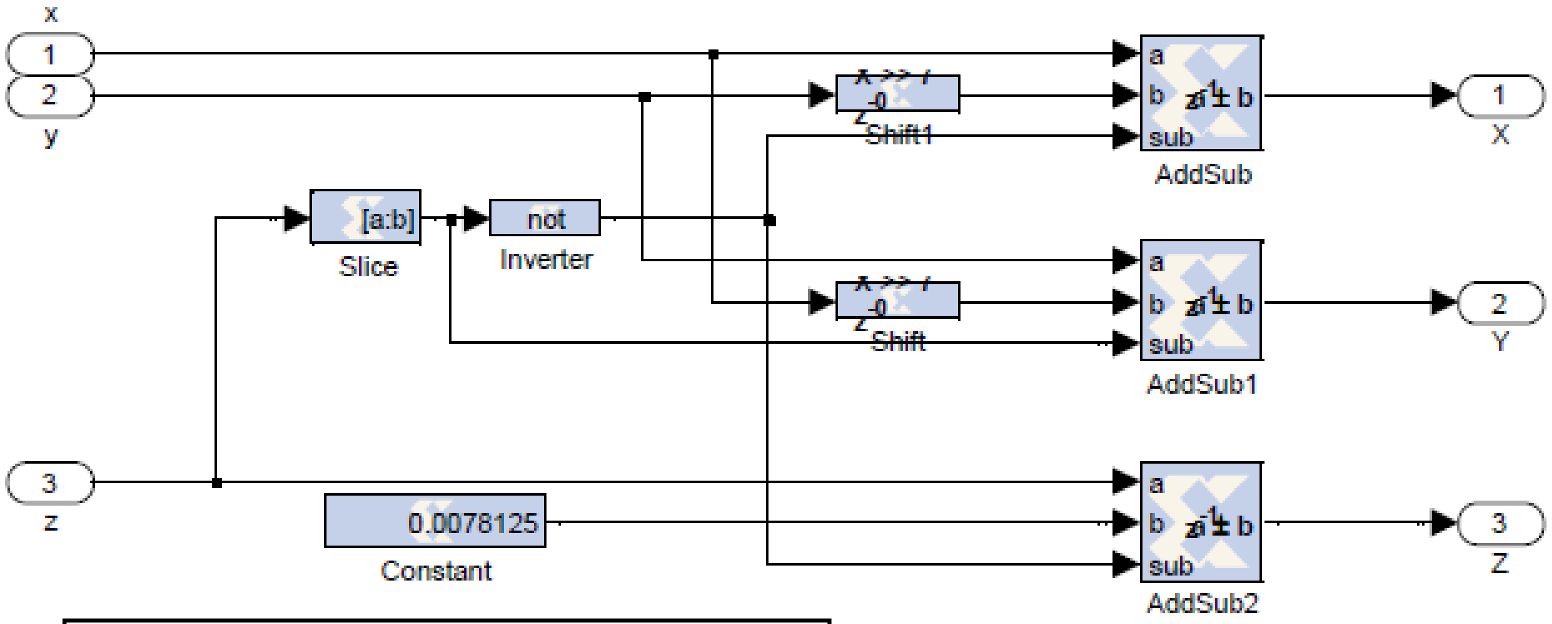


CORDIC – First Micro-rotation and Decision Logic



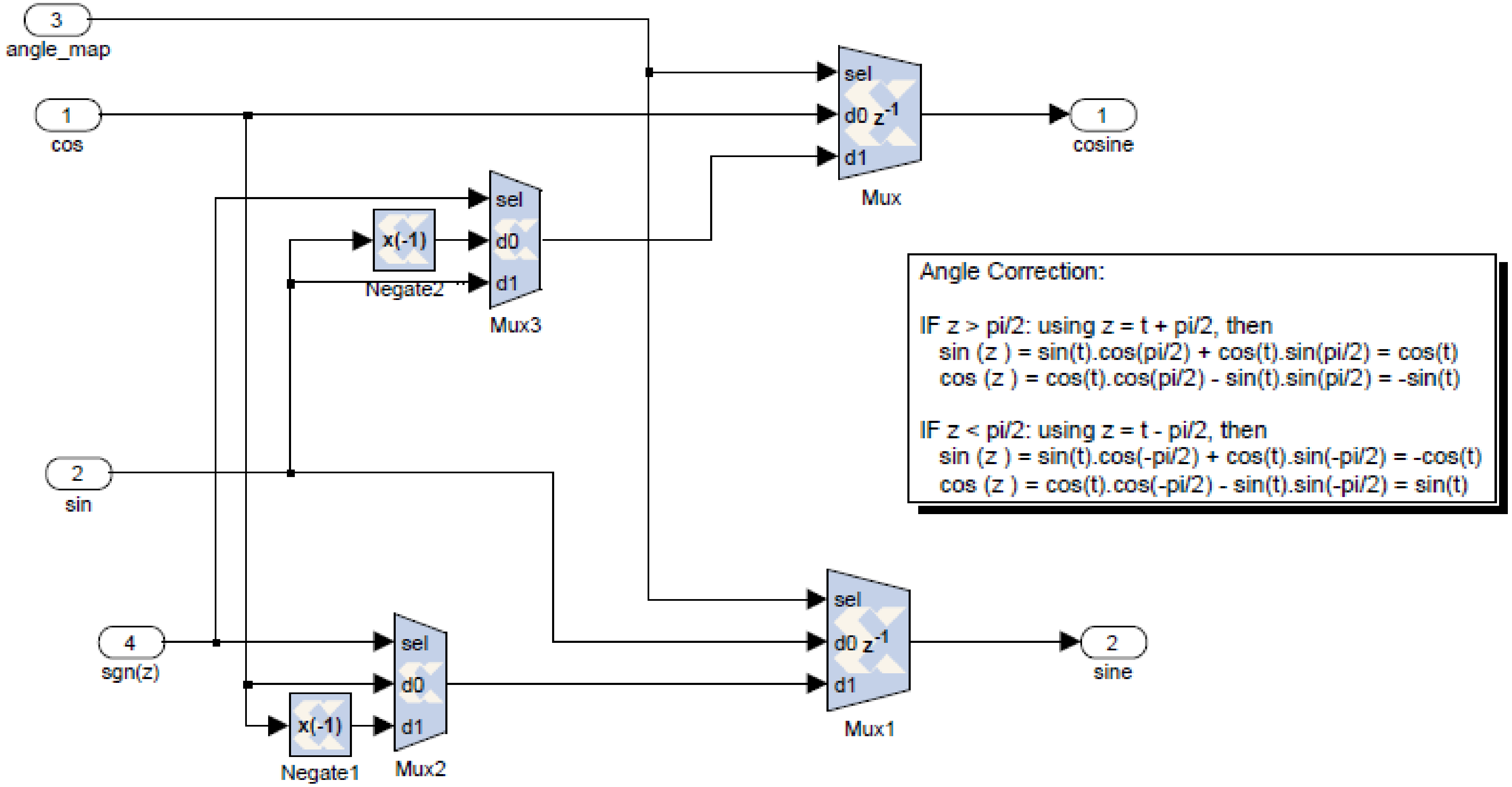
X = x + y if z < 0, otherwise
= x - y
Y = y - x if z < 0, otherwise
= y + x
Z = z + atan(1/2^i) if z < 0, otherwise
= z - atan(1/2^i)

CORDIC - 8th Final Rotation in Example



X = x + y if z < 0, otherwise
= x - y
Y = y - x if z < 0, otherwise
= y + x
Z = z + atan(1/2^i) if z < 0, otherwise
= z - atan(1/2^i)

CORDIC – Final Correction Step



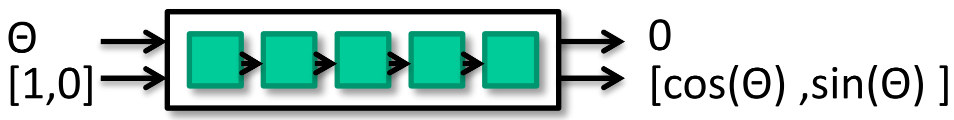
Modes of CORDIC algorithm

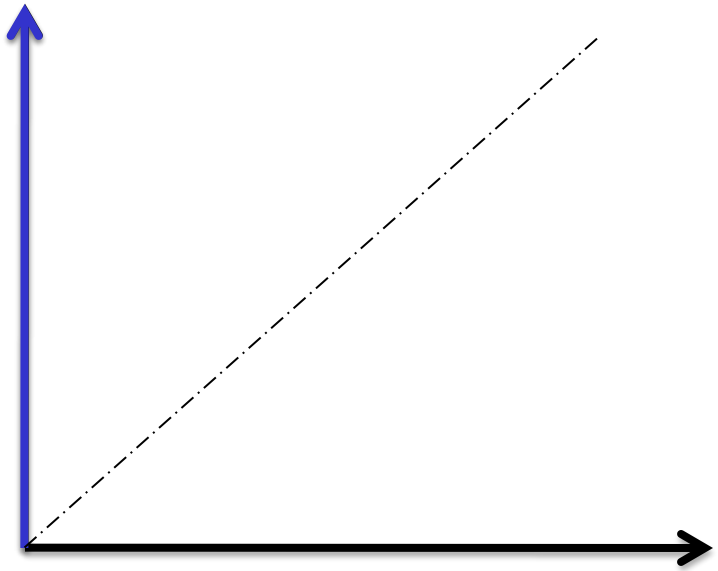
- Cheap iterative algorithm – iterative successive unitary micro-rotations
- Modes useful to us:
  - $\Rightarrow$ ATAN – “find” the angle between a pair of x and y values
  - $\Rightarrow$ COS/SIN – then able to also “rotate” a vector by an angle

  - $\Rightarrow$ Scale Factor Correction or Compensation complicates the use of CORDIC.
  - $\Rightarrow$ Model Composer modules can apply compensation iterations
  - $\Rightarrow$ If done in Vitis HLS then extra iterations need to be coded
  - $\Rightarrow$ The output x and y variables will need correction for vector rotation
    - Exceptions occur if preloading for COS/SIN or ATAN

Review of CORDIC algorithm

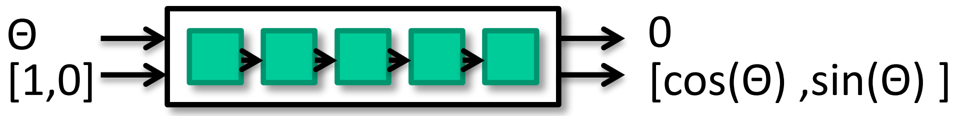
- COS/SIN Block
  - ⇒ Input: Θ, Initial vector [1,0]
  - ⇒ Output: sin(Θ), cos(Θ)
  - ⇒ Apply successively smaller unitary micro-rotations

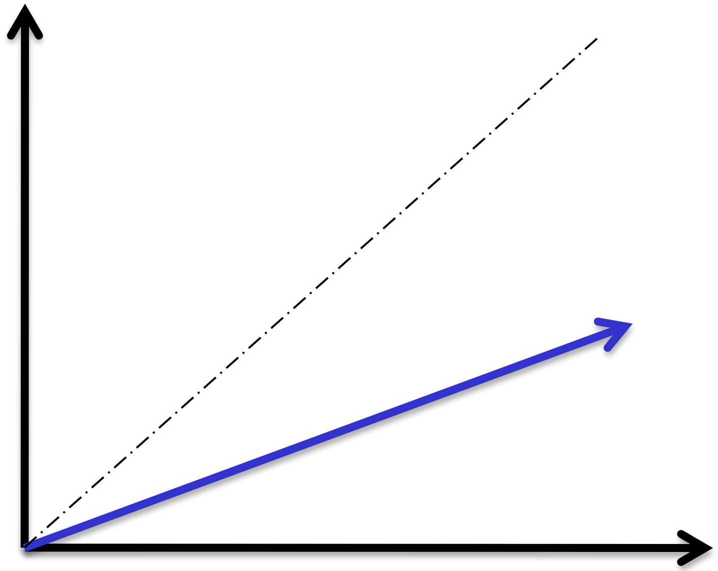




Review of CORDIC algorithm

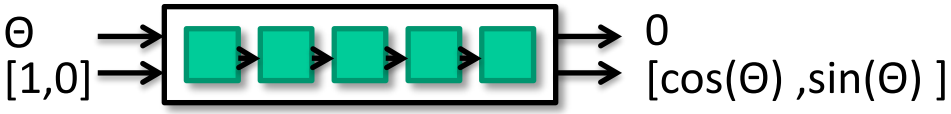
- COS/SIN Block
  - ⇒ Input: Θ, Initial vector [1,0]
  - ⇒ Output: sin(Θ), cos(Θ)
  - ⇒ Apply successively smaller unitary micro-rotations

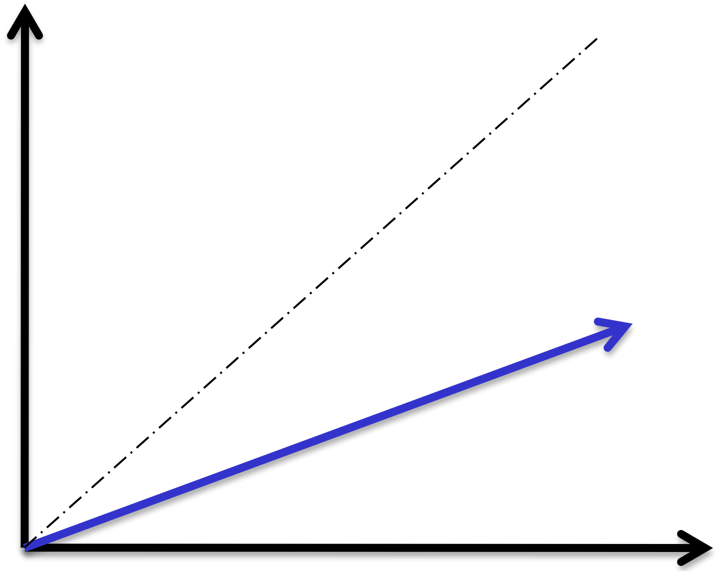




Review of CORDIC algorithm

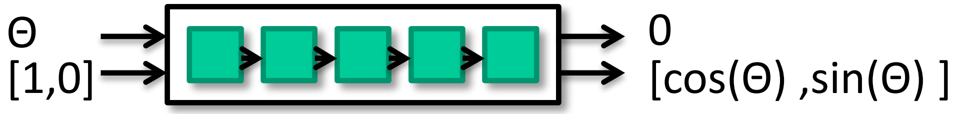
- COS/SIN Block
  - ⇒ Input: Θ, Initial vector [1,0]
  - ⇒ Output: sin(Θ), cos(Θ)
  - ⇒ Apply successively smaller unitary micro-rotations

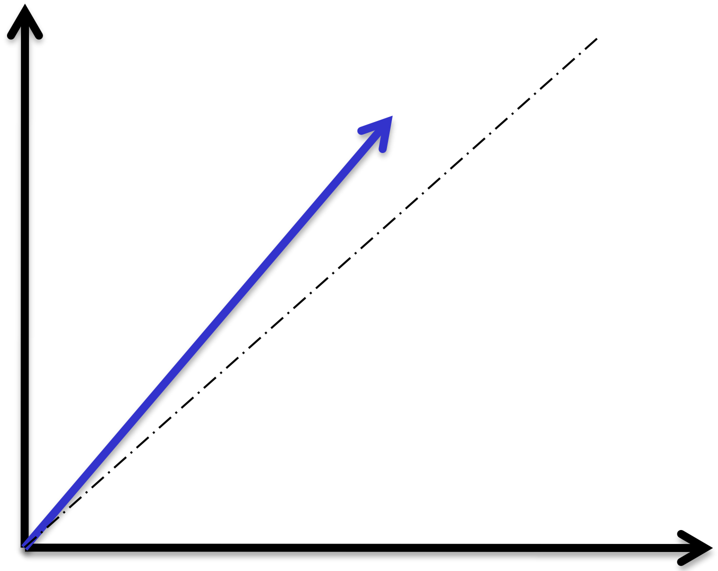




Review of CORDIC algorithm

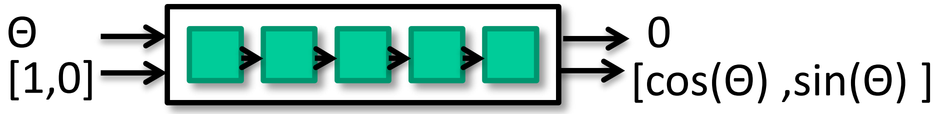
- COS/SIN Block
  - ⇒ Input: Θ, Initial vector [1,0]
  - ⇒ Output: sin(Θ), cos(Θ)
  - ⇒ Apply successively smaller unitary micro-rotations





Review of CORDIC algorithm

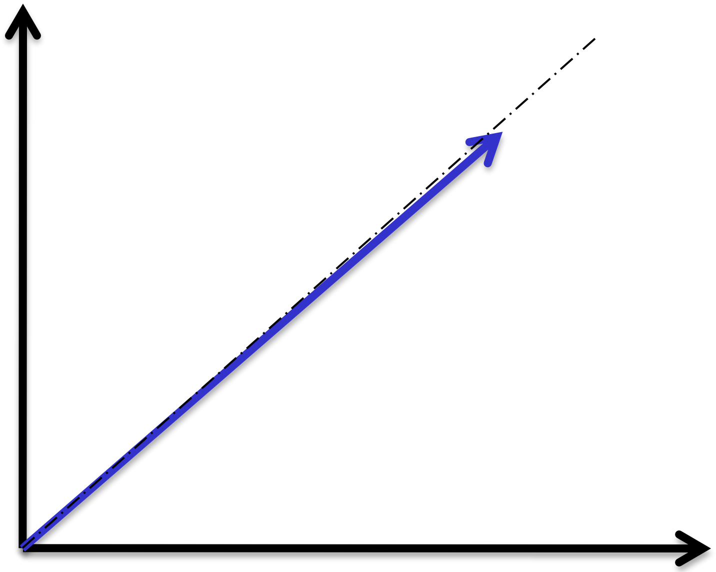
- COS/SIN Block
  - ⇒ Input: Θ, Initial vector [1,0]
  - ⇒ Output: sin(Θ), cos(Θ)
  - ⇒ Apply successively smaller unitary micro-rotations



$$v_o = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

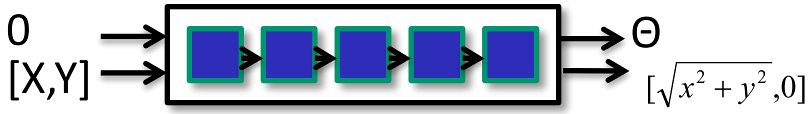
$$R_i = \begin{pmatrix} \cos\gamma_i & -\sin\gamma_i \\ \sin\gamma_i & \cos\gamma_i \end{pmatrix}$$

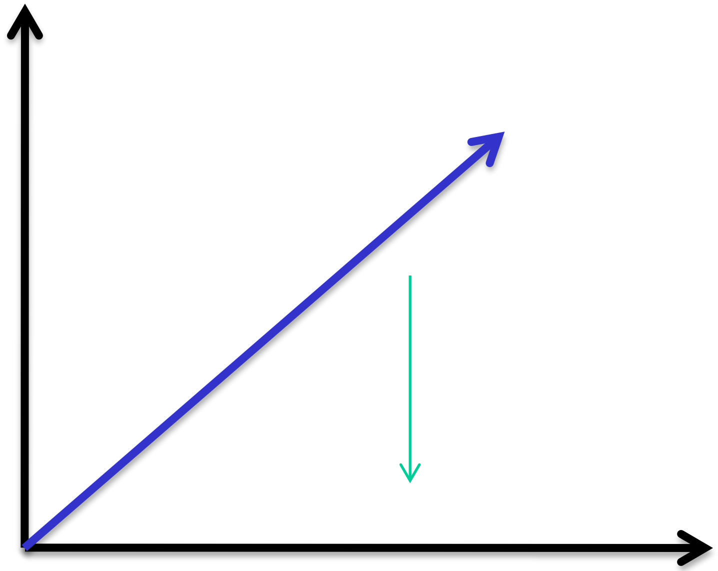
$$v_{i+1} = R_i v_i = \cos\gamma_i \begin{pmatrix} 1 & -\sigma_i \tan\gamma_i \\ \sigma_i \tan\gamma_i & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$



Review of CORDIC algorithm

- ATAN Block
  - ⇒ Input: Θ, Initial vector [X,Y], initial angle 0
  - ⇒ Output: Θ, [R,0]
  - ⇒ Apply successively smaller unitary micro-rotations
  - ⇒ The vector magnitude or radius will need to be scale factor compensated





Discussion on Applications of CORDIC

- CORDIC can be used in isolated Sin and Cos calculations
- CORDIC use in graphics for Rotations or images
- CORDIC use in solving systems of linear equations.
- Can solve y = Ax + b many ways:
  - ⇒ Gaussian elimination most basic
    - Problems with division and roundoff
    - Especially when matrix is ill-formed.
    - Pivoting is sometimes used.
    - CMOR courses covers much of this
- Orthogonal rotations are “better” to preserve norm.
  - ⇒ Less fixed-point issues -> QR Decomposition

CORDIC for Givens Rotations

- Givens Rotations
  - ⇒ Conceptually vector rotation
  - ⇒ Based on Sine and Cosine
  - ⇒ Good numerical properties – Norm preserving

- Applied to Matrix Factorization to solve systems of Linear Equations
  - ⇒ QRD, SVD, Eigenvalue Decomposition
  - ⇒ Focus on QRD which factors A into Q and R
    - R is Right upper triangular
    - Q is Orthogonal and based on collection of sin, cos

Givens Rotations Build on CORDIC

$$G(i,k,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

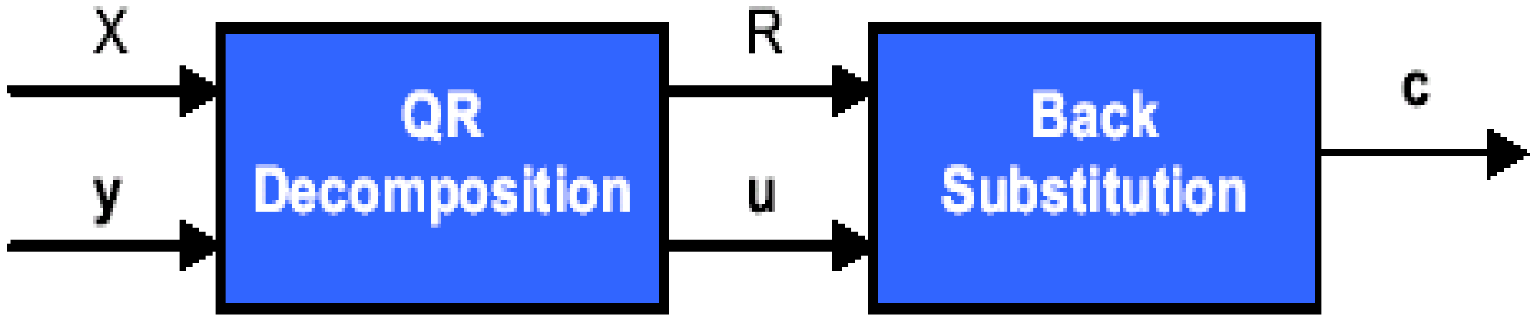
Givens Rotations

$$A = \begin{pmatrix} 12 & -51 & 4 \\ 6 & 167 & -68 \\ -4 & 24 & -41 \end{pmatrix}.$$

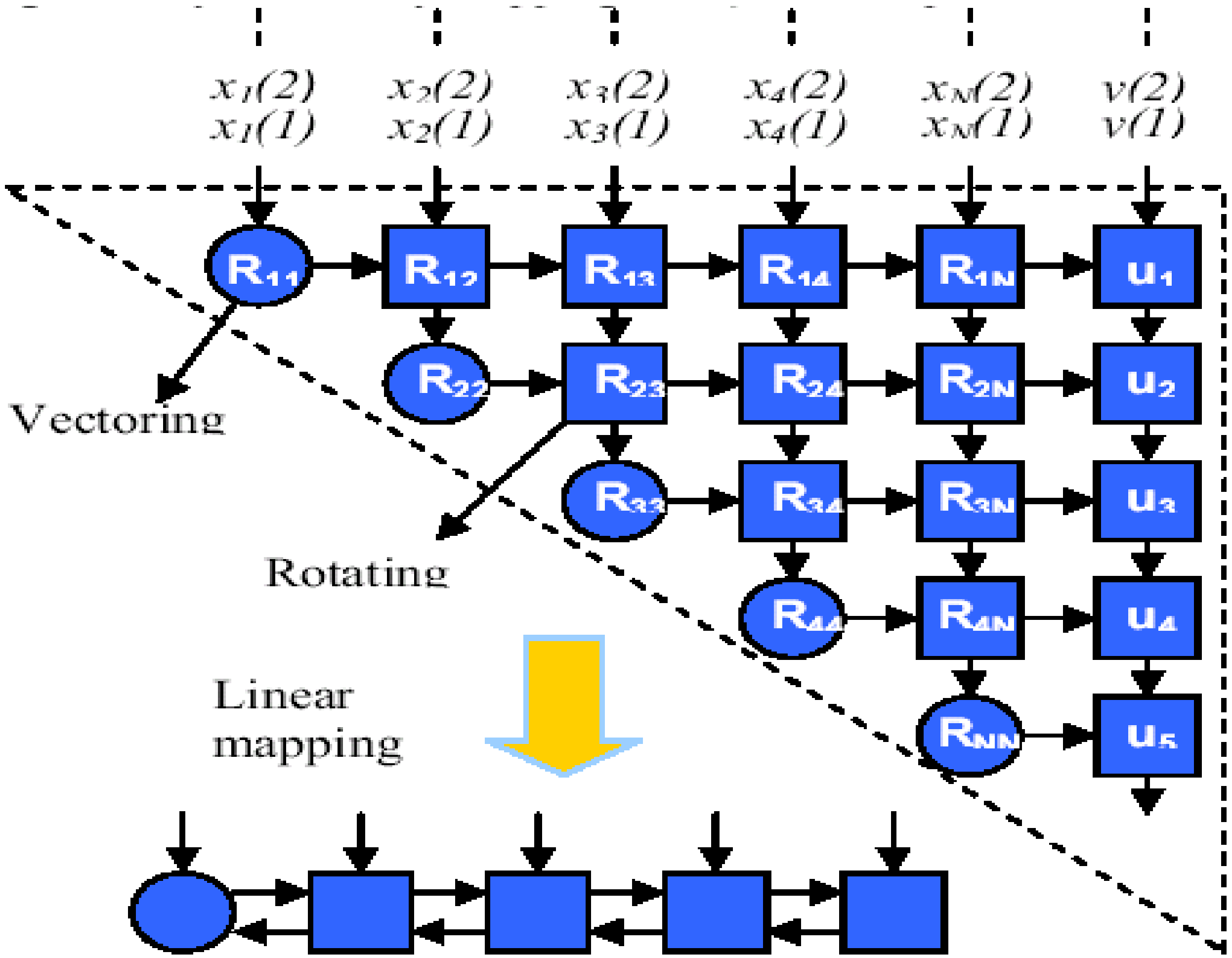
$$\begin{aligned} G_1 &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{pmatrix} \\ &\approx \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.83205 & -0.55470 \\ 0 & 0.55470 & 0.83205 \end{pmatrix} \end{aligned}$$

QRD to Solve Linear System of Equations

$$X\mathbf{c} = \mathbf{y} + \mathbf{e}$$



QRD Systolic Array



QRD CORDIC Papers on Canvas

- FPGA based paper – Altera Group
- A High Throughput Systolic Design for QR Algorithm
- FPGA Implementation of Matrix Inversion using QRD-RLS Algorithm
- Triangular Systolic Array with Reduced Latency for QR-decomposition of Complex Matrices
- High-Throughput QR Decomposition for MIMO Detection of OFDM Systems

Readings on QR Decomposition Arrays and CORDIC

- W09_1982_Ahmed_Morf_Delosme_01653828.pdf
- W09_1984_Bojanczyk_Brent_Kung_SIAM.pdf
- W09_1988_JPDC_CORIDC_SVD_Cavallaro.pdf
- W09_1993_ISCAS_QR_Systolic00693005[1].pdf
- W09_2005_Asilomar_QRD_RLS_Karkooti.pdf
- W09_2006_ISCAS_QRD_Tri_complex.pdf
- W09_2010_ISCAS_QRD_MIMO_05537358.pdf

- In Canvas Papers_Readings
  - ⇒ Will be helpful background for Projects 4 and 5

Next Lecture

- More on CORDIC Arithmetic
- Discussion of Project 4 on CORDIC Arithmetic
- Begin discussion of QR Decomposition using CORDIC