Lecture

7

ECE216B

CORDIC, Divider, Square Root

Prof. Dejan Marković

ee216b@gmail.com

Agenda

- **Iterative algorithms**
  - CORDIC
  - Division, square root
- **Discussion topics**
  - Algorithms, baseline architecture
  - Convergence analysis
  - The choice of initial condition
- **SVD chip example**

CORDIC

To perform the following transformation

$$y(t) = y_R + j \cdot y_I \rightarrow |y| \cdot e^{j\varphi}$$

and the inverse, we use the CORDIC algorithm

CORDIC:

COordinate Rotation DIgital Computer

CORDIC: Idea

Use rotations to implement a variety of functions

Examples:

$$x + j \cdot y \Leftrightarrow |\sqrt{x^2 + y^2}\,| e^{j \cdot tan^{-1}(y/x)}$$

$$z = \sqrt{x^2 + y^2} \qquad z = \cos(y / x) \qquad z = \tan(y / x)$$

$$z = x / y \qquad z = \sin(y / x) \qquad z = \sinh(y / x)$$

$$z = tan^{-1}(y / x) \qquad z = cos^{-1}(y)$$

CORDIC: How to Do It?

- Start with general rotation by φ

$$x' = x \cdot \cos(\varphi) - y \cdot \sin(\varphi)$$
$$y' = y \cdot \cos(\varphi) + x \cdot \sin(\varphi)$$

$$x' = \cos(\varphi) \cdot [x - y \cdot \tan(\varphi)]$$
$$y' = \cos(\varphi) \cdot [y + x \cdot \tan(\varphi)]$$

- The trick is to only do rotations by values of $\tan(\varphi)$ which are powers of 2

CORDIC: An Iterative Process

- To rotate to any arbitrary angle, we do a sequence of rotations to get to that value

Rotation Number

φ	$\tan(\varphi)$	k	i
45°	1	1	0
26.565°	2^{-1}	2	1
14.036°	2^{-2}	3	2
7.125°	2^{-3}	4	3
3.576°	2^{-4}	5	4
1.790°	2^{-5}	6	5
0.895°	2^{-6}	7	6

Basic CORDIC Iteration

Gain Direction Angle

$$x_{i+1} = (K_i) \cdot [x_i - y_i \cdot d_i \cdot 2^{-i}]$$
$$y_{i+1} = (K_i) \cdot [y_i + x_i \cdot d_i \cdot 2^{-i}]$$

$$K_i = \cos(\tan^{-1}(2^{-i})) = 1/(1 + 2^{-2i})^{0.5}$$

$$d_i = \pm 1 \text{ (rotate by } \pm\varphi)$$

- **If we don't multiply by K_i we get a gain error which is independent of the direction of the rotation**
  - The error converges to 0.61
  - May not need to compensate for it

Basic CORDIC Iteration

Gain Direction Angle

$$x_{i+1} = (K_i) \cdot [x_i - y_i \cdot d_i \cdot 2^{-i}]$$
$$y_{i+1} = (K_i) \cdot [y_i + x_i \cdot d_i \cdot 2^{-i}]$$

$$K_i = \cos(\tan^{-1}(2^{-i})) = 1/(1 + 2^{-2i})^{0.5}$$

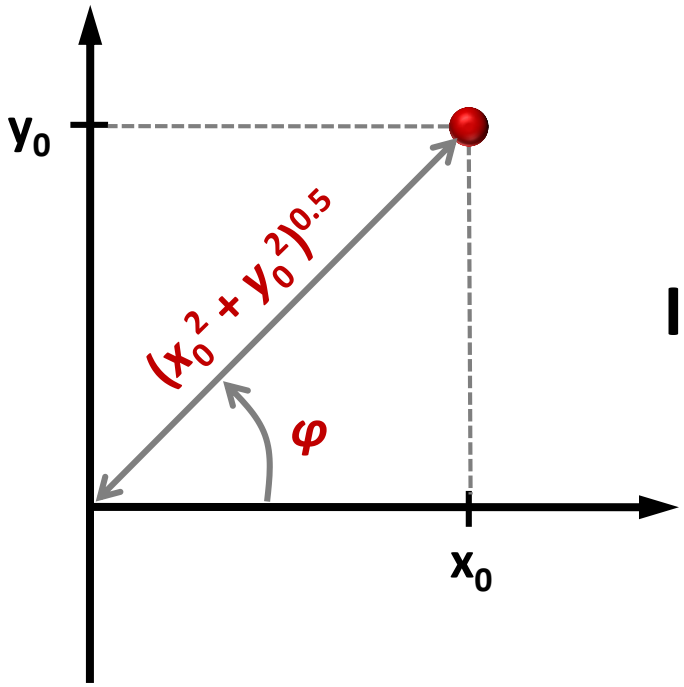
$d_i = \pm 1$ (rotate by $\pm\varphi$)

- We can also accumulate the rotation angle:

$$z_{i+1} = z_i - d_i \cdot \tan^{-1}(2^{-i})$$

Example 8.1: Rectangular-to-Polar Conversion

Initial vector is described by x_0 and y_0 coordinates

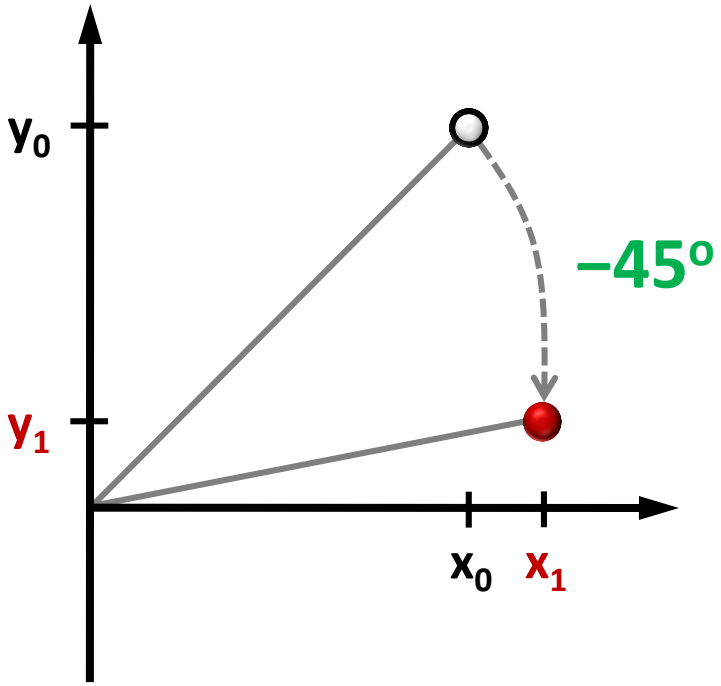


Initial vector

➪ **Find φ and $(x_0^2 + y_0^2)^{0.5}$**

Step 1: Check the Angle / Sign of y_0

- If positive, rotate by –45°
- If negative, rotate by +45°



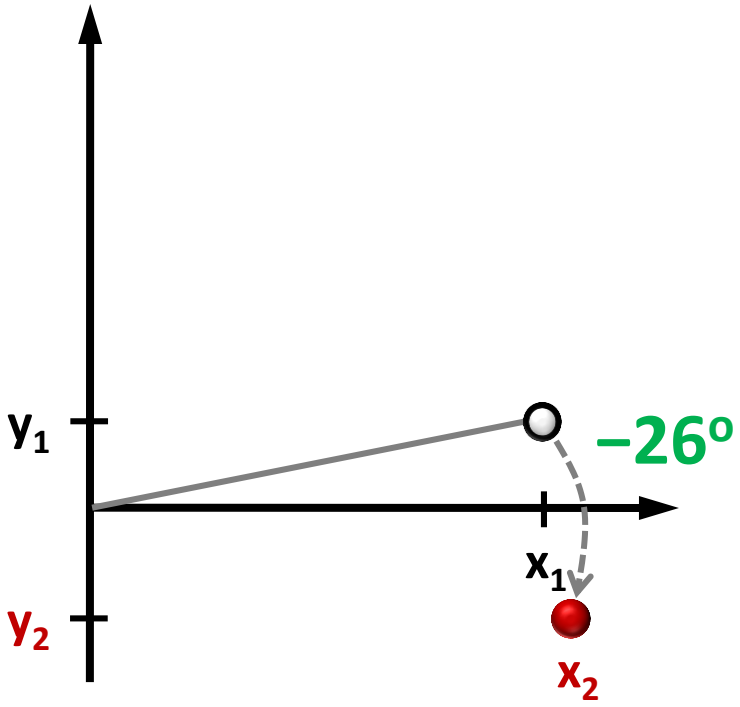
$d_1 = -1$ $(y_0 > 0)$

$x_1 = x_0 + y_0$

$y_1 = y_0 - x_0$

Step 2: Check the Sign of y_1

- If positive, rotate by –26.57°
- If negative, rotate by +26.57°



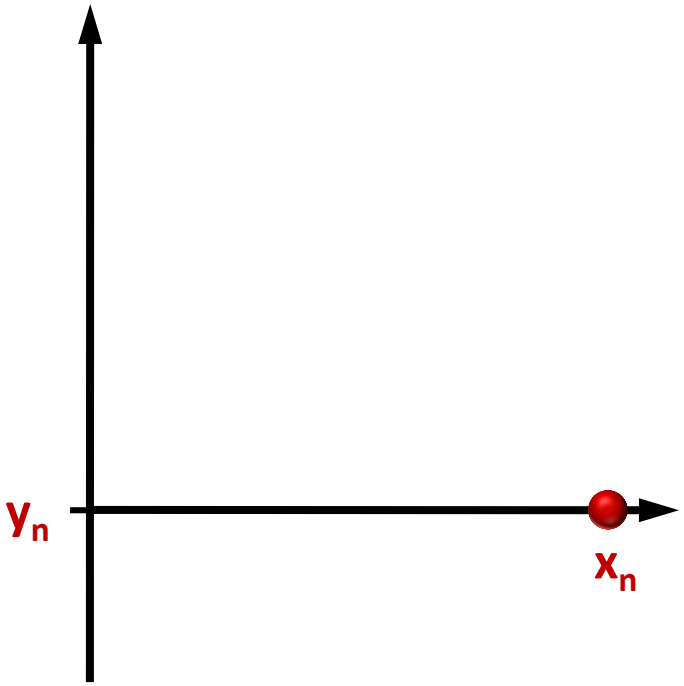
$d_2 = -1$ $(y_1 > 0)$

$x_2 = x_1 + y_1 / 2$

$y_2 = y_1 - x_1 / 2$

Repeat Step 2 for Each Rotation *k*

- Until $y_n = 0$



$$y_n = 0$$

$$x_n = A_n \cdot (x_0^2 + y_0^2)^{0.5}$$

accumulated gain

The Gain Factor

- **Gain accumulation** (when you don't multiply by K_i)

  $G_0 = 1$

  $G_0G_1 = 1.414$

  $G_0G_1G_2 = 1.581$

  $G_0G_1G_2G_3 = 1.630$

  $G_0G_1G_2G_3G_4 = 1.642$

- **So, start with x_0, y_0; end up with:**

  Shift & adds of x_0, y_0

  $z_4 = 71^o$

  $(x_0^2 + y_0^2)^{0.5} = 1.642$ (…)

- **We did the rectangular-to-polar coordinate conversion**

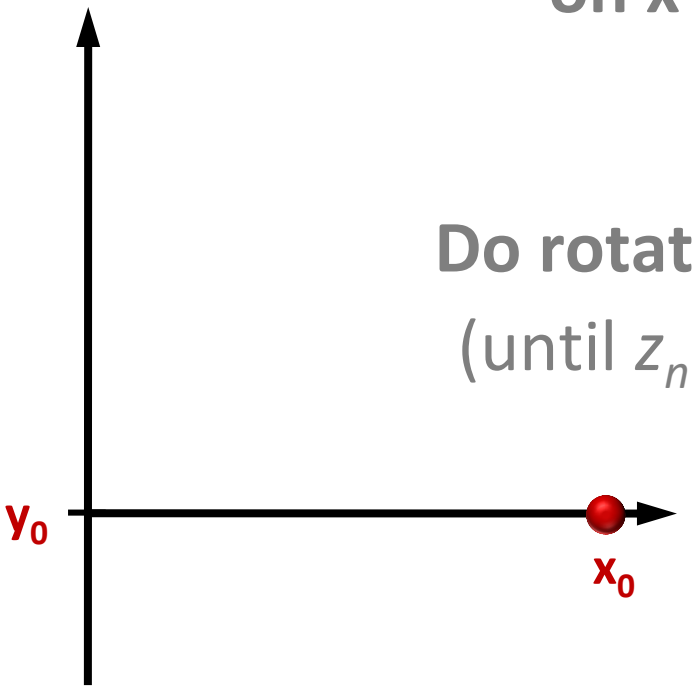
Polar-to-Rectangular Conversion

Start with vector on x-axis

$A = |A| \cdot e^{j\varphi}$

$x_0 = |A|$

$y_0 = 0, z_0 = \varphi$



Do rotations (until $z_n = 0$)

$z_i < 0, d_i = -1$

$z_i > 0, d_i = +1$

$z_{i+1} = z_i - d_i \cdot \tan^{-1}(2^{-i})$

CORDIC Algorithm

$$x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i}$$
$$y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}$$
$$z_{i+1} = z_i - d_i \cdot \tan^{-1}(2^{-i})$$

$$d_i = \begin{cases} -1, & z_i < 0 \\ +1, & z_i > 0 \end{cases}$$

$$d_i = \begin{cases} -1, & y_i > 0 \\ +1, & y_i < 0 \end{cases}$$

Rotation mode

(rotate by specified angle)

Minimize residual angle

$$x_n = A_n \cdot [x_0 \cdot \cos(z_0) - y_0 \cdot \sin(z_0)]$$
$$y_n = A_n \cdot [y_0 \cdot \cos(z_0) + x_0 \cdot \sin(z_0)]$$
$$z_n = 0$$

Vectoring mode

(align with the *x*-axis)

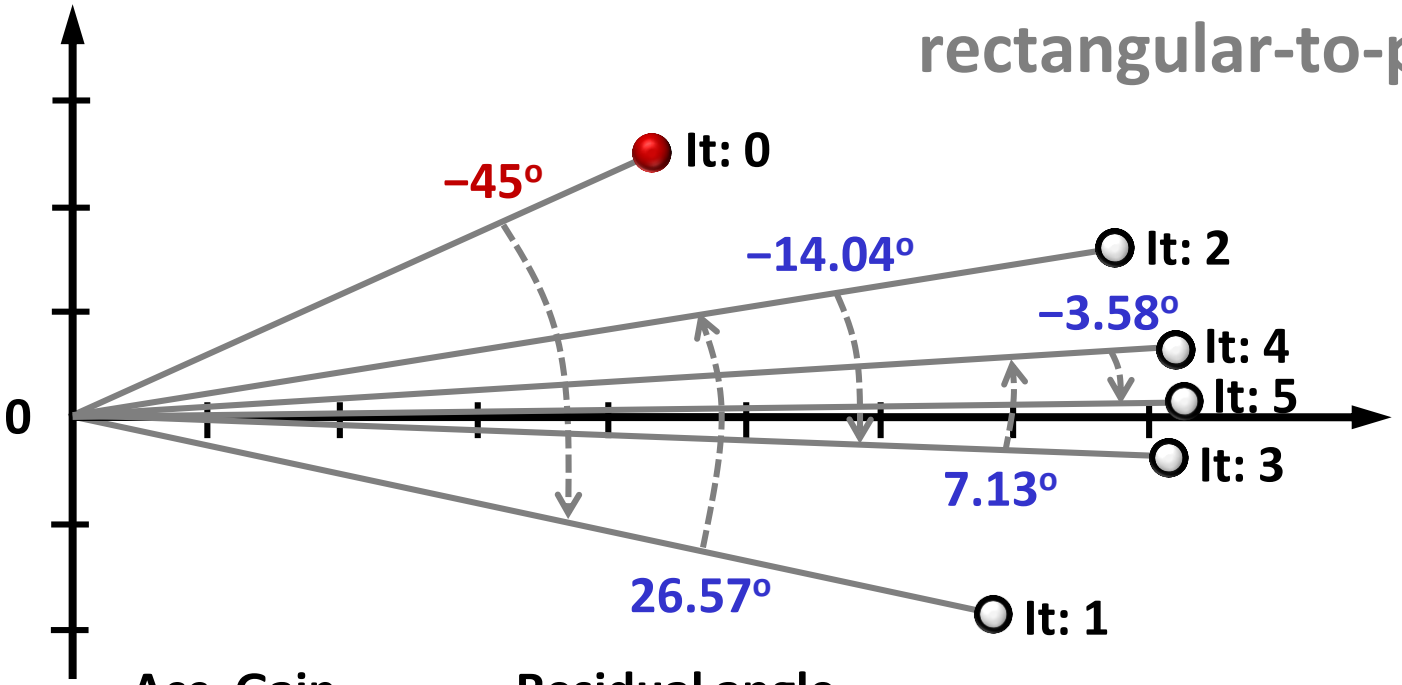
Minimize *y* component

$$x_n = A_n \cdot (x_0^2 + y_0^2)^{0.5}$$
$$y_n = 0$$
$$z_n = z_0 + \tan^{-1}(y_0/x_0)$$

$$A_n = \prod^{n}(1+2^{-2i})^{0.5} \rightarrow 1.647$$

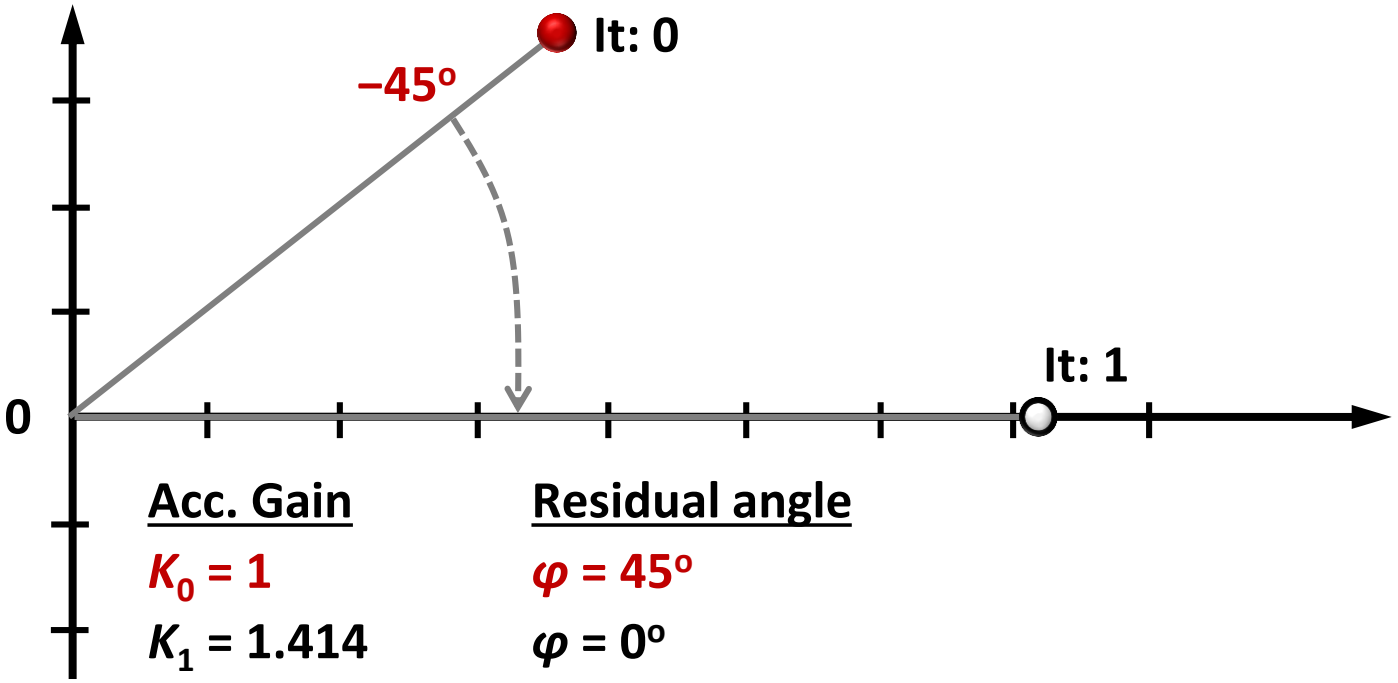
Example 8.2: Another Vectoring Example

rectangular-to-polar



Acc. Gain	Residual angle
$K_0 = 1$	$\varphi = 30^\circ$
$K_1 = 1.414$	$\varphi = -15^\circ$
$K_2 = 1.581$	$\varphi = 11.57^\circ$
$K_3 = 1.630$	$\varphi = -2.47^\circ$
$K_4 = 1.642$	$\varphi = 4.65^\circ$
$K_5 = 1.646$	$\varphi = 1.08^\circ$
Etc.	Etc.

Vectoring: Best-Case Convergence

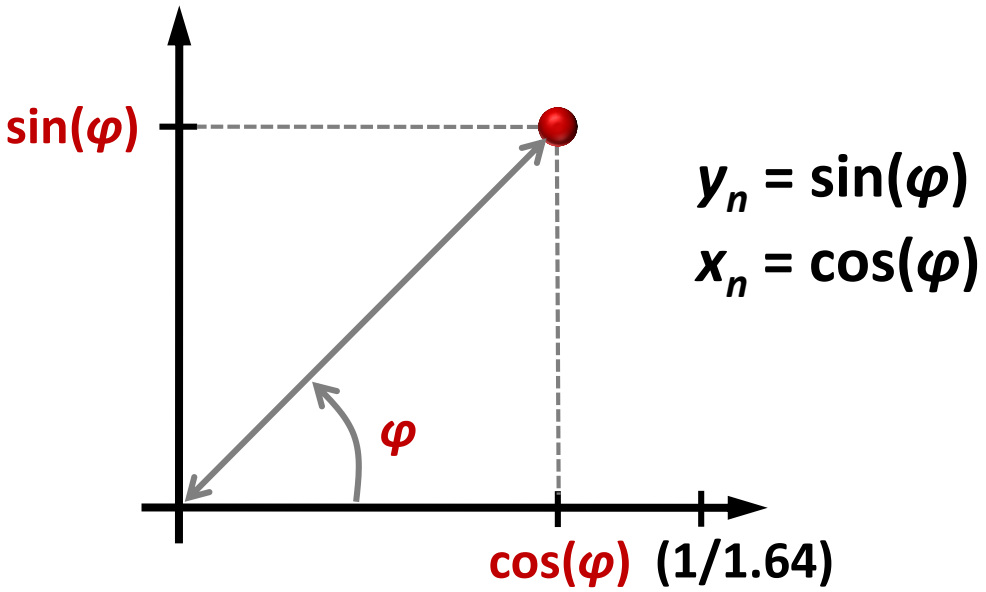


Acc. Gain	Residual angle
$K_0 = 1$	$\varphi = 45^\circ$
$K_1 = 1.414$	$\varphi = 0^\circ$
$K_2 = 1.581$	$\varphi = 0^\circ$
$K_3 = 1.630$	$\varphi = 0^\circ$
$K_4 = 1.642$	$\varphi = 0^\circ$
$K_5 = 1.646$	$\varphi = 0^\circ$
Etc.	

In the best case ($\varphi = 45^\circ$), we can converge in 1 iteration

Calculating Sine and Cosine

- Start with $x_0 = 1/1.64$, $y_0 = 0$
- Rotate by φ



$y_n = \sin(\varphi)$

$x_n = \cos(\varphi)$

Functions

Rotation mode

sin/cos

$z_0 = \text{angle}$

$y_0 = 0,\ x_0 = 1/A_n$

$x_n = A_n \cdot x_0 \cdot \cos(z_0)$

$y_n = A_n \cdot x_0 \cdot \sin(z_0)$

$(=1)$

Vectoring mode

$\tan^{-1}$

$z_0 = 0$

$z_n = z_0 + \tan^{-1}(y_0/x_0)$

Vector/Magnitude

$x_n = A_n \cdot (x_0^2 + y_0^2)^{0.5}$

Polar → Rectangular

$x_n = r \cdot \cos(\varphi)$

$y_n = r \cdot \sin(\varphi)$

$x_0 = r$

$z_0 = \varphi$

$y_0 = 0$

Rectangular → Polar

$r = (x_0^2 + y_0^2)^{0.5}$

$\varphi = \tan^{-1}(y_0/x_0)$

CORDIC Divider

- **To do a divide, change CORDIC rotations to a linear function calculator**
  - Provided that $x_0 > y_0/2$

$$x_{i+1} = x_i - \boxed{0} \cdot y_i \cdot d_i \cdot 2^{-i} = x_i$$

$$y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}$$

$$z_{i+1} = z_i - d_i \cdot \boxed{(2^{-i})}$$

Generalized CORDIC

$$x_{i+1} = x_i - m \cdot y_i \cdot d_i \cdot 2^{-i}$$

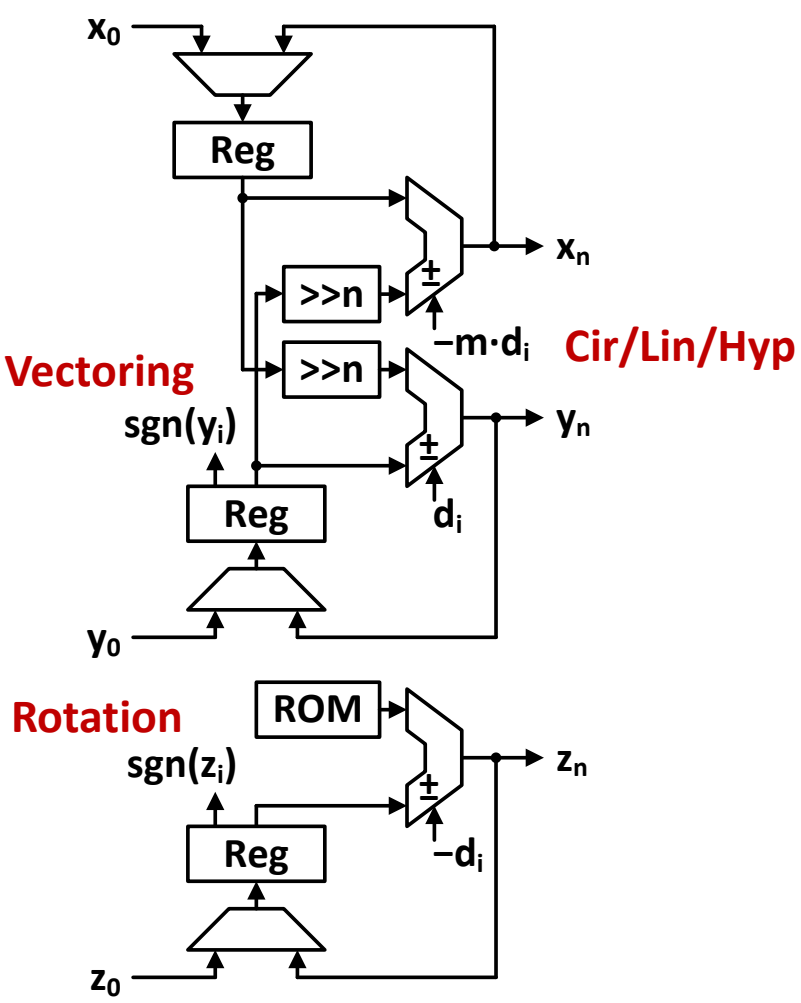
$$y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}$$

$$z_{i+1} = z_i - d_i \cdot e_i$$

d_i	
Rotation	**Vectoring**
$d_i = -1,\ z_i < 0$ $d_i = +1,\ z_i > 0$	$d_i = -1,\ y_i > 0$ $d_i = +1,\ y_i < 0$
$\text{sign}(z_i)$	$-\text{sign}(y_i)$

Mode	m	e_i
Circular	+1	$\tan^{-1}(2^{-i})$
Linear	0	2^{-i}
Hyperbolic	−1	$\tanh^{-1}(2^{-i})$

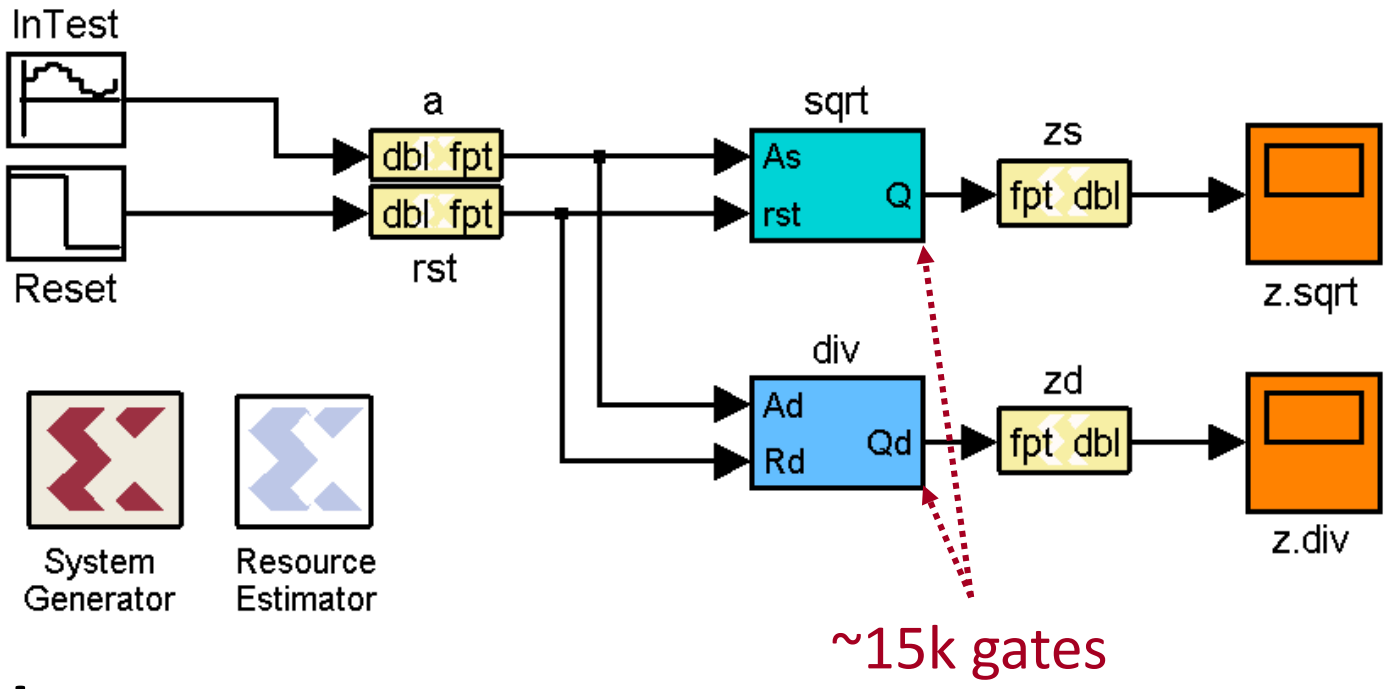
An FPGA Implementation [1]



- **Three difference equations directly mapped to hardware**
- **The decision d_i is driven by the sign of the *y* or *z* register**
  - **Vectoring:** $d_i = -sign(y_i)$
  - **Rotation:** $d_i = sign(z_i)$
- **The initial values loaded via muxes**
- **On each clock cycle**
  - Register values are passed through shifters and add/sub and the values placed in registers
  - The shifters are modified on each iteration to cause the desired shift (state machine)
  - Elementary angle stored in ROM
- **Last iteration: results read from reg**

[1] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *Proc. Int. Symp. Field Programmable Gate Arrays*, Feb. 1998, pp. 191-200.

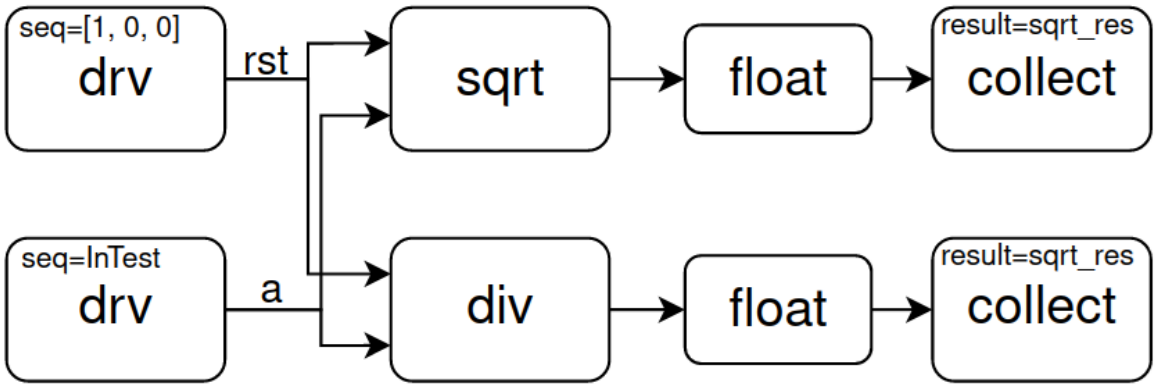
Iterative Sqrt and Division*



- **Inputs:**
  - **a (14 bits), reset (active high)**
- **Outputs:**
  - **zs (16 bits), zd (16 bits)** **Total: 32 bits**

*C. Ramamoorthy, J. Goodman, and K. Kim, "Some Properties of Iterative Square-Rooting Methods Using High-Speed Multiplication," *IEEE Trans. Computers,* vol. C-21, no. 8, pp. 837–847, Aug. 1972.

Iterative Sqrt and Division: PyGears Model



Output precision (16 bits) specified in **sqrt** and **div** "gears"

```
Tinput = Ufixp[6, 14]
InTest = [0, 1, 2]
sqrt_res = []
div_res = []
a = drv(t=Tinput, seq=InTest)
rst = drv(t=Bool, seq=[1, 0, 0])
a | sqrt(rst) | float | collect(result=sqrt_res)
a | div(rst) | float | collect(result=div_res)
```

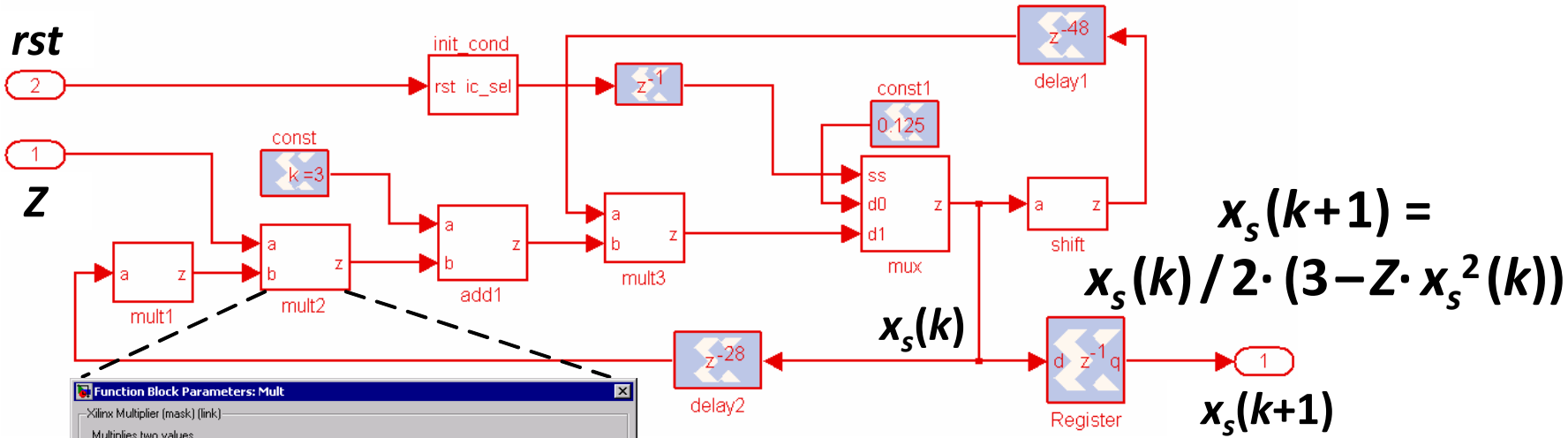
format of input a

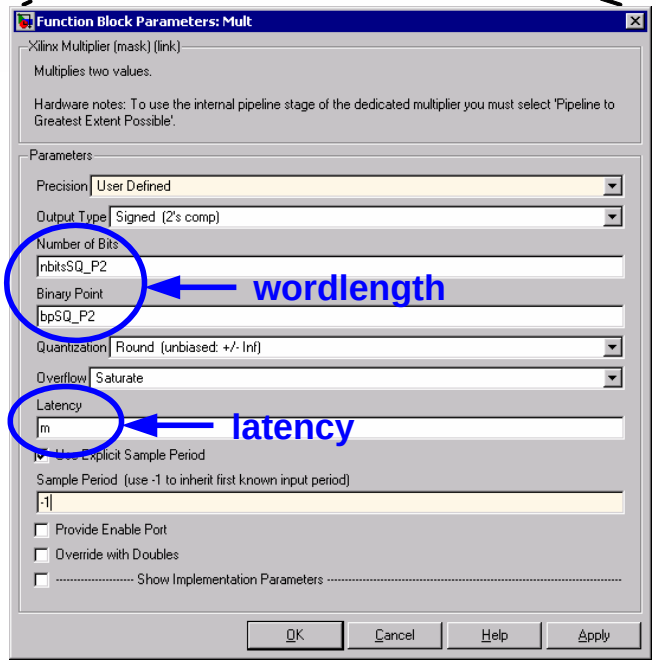
PyGears

Zd is here

Zd in float

Iterative 1/sqrt(Z): Simulink XSG Model

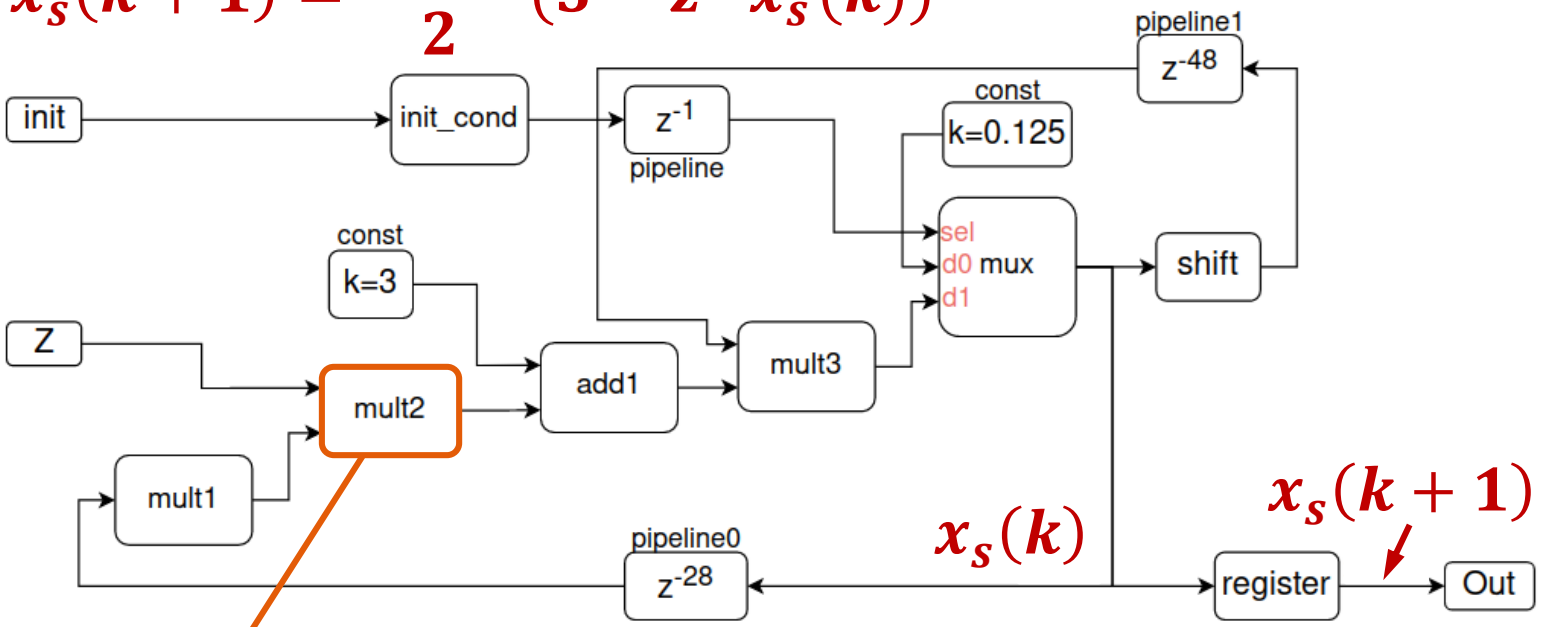




- **User defined parameters**
  - Wordlength (#bits, binary pt)
  - Quantization, overflow
  - Latency, sample period
- **The choice of initial condition**
  - Determines # iterations
  - and convergence…

Iterative 1/sqrt(Z): PyGears Model

$$x_s(k+1) = \frac{x_s(k)}{2}(3 - z \cdot x_s^2(k))$$




```
m2 = mult_dsp(z, m1,
              t=Fixp[6, 14],
              overflow=Overflow.SATURATE,
              quantization=Quantization.ROUND,
              latency=4)
```

output format

multiplier latency

PyGears

Iterative 1/sqrt(*Z*): Complete PyGears Model

```
@gear
def inv_sqrt(z, init, *, dtype, my_latency):
    x = Intf(dtype)
    x_d = x | pipeline(length=28, feedback=True)

    m1 = mult_dsp(x_d, x_d,
                  t=dtype,
                  overflow=Overflow.SATURATE,
                  quantization=Quantization.ROUND,
                  latency=my_latency)

    m2 = mult_dsp(z, m1,
                  t=dtype,
                  overflow=Overflow.SATURATE,
                  quantization=Quantization.ROUND,
                  latency=my_latency)
```

Continued on the next slide

mult_dsp help:

https://github.com/bogdanvuk/pygears-dsp/blob/master/pygears_dsp/lib/basic_blocks.py

Iterative 1/sqrt(*Z*): Complete PyGears Model

Continued from the previous slide

```
    a1 = add_sub_dsp(dtype(3), m2,
                     operation=Operation.SUB,
                     t=dtype,
                     overflow=Overflow.SATURATE,
                     quantization=Quantization.TRUNCATE,
                     latency=my_latency)

    m3 = mult_dsp(a1, (x >> 1) | pipeline(length=48,
feedback=True),
                  t=dtype,
                  overflow=Overflow.SATURATE,
                  quantization=Quantization.ROUND,
                  latency=my_latency)

    x |= mux_dsp(init_cond(init), dtype(0.125), m3)
    return x | dreg
```

Quadratic Convergence: 1/sqrt(*N*)

$$x_s(k+1) = \frac{x_s(k)}{2} \cdot (3 - N \cdot x_s(k)^2) \qquad x_s(k) \to \frac{1}{\sqrt{N}}\bigg|_{k\to\infty}$$

$$y_s(k+1) = \frac{y_s(k)}{2} \cdot (3 - y_s(k)^2) \qquad y_s(k) \to 1\big|_{k\to\infty}$$

$$e_s(k) = y_s(k) - 1$$

$$e_s(k+1) = -\frac{1}{2} \cdot e_s(k)^2 \cdot (3 + e_s(k))$$

Quadratic Convergence: 1/*N*

$$x_d(k+1) = x_d(k) \cdot (2 - N \cdot x_d(k)) \qquad x_d(k) \to \left. \frac{1}{N} \right|_{k \to \infty}$$

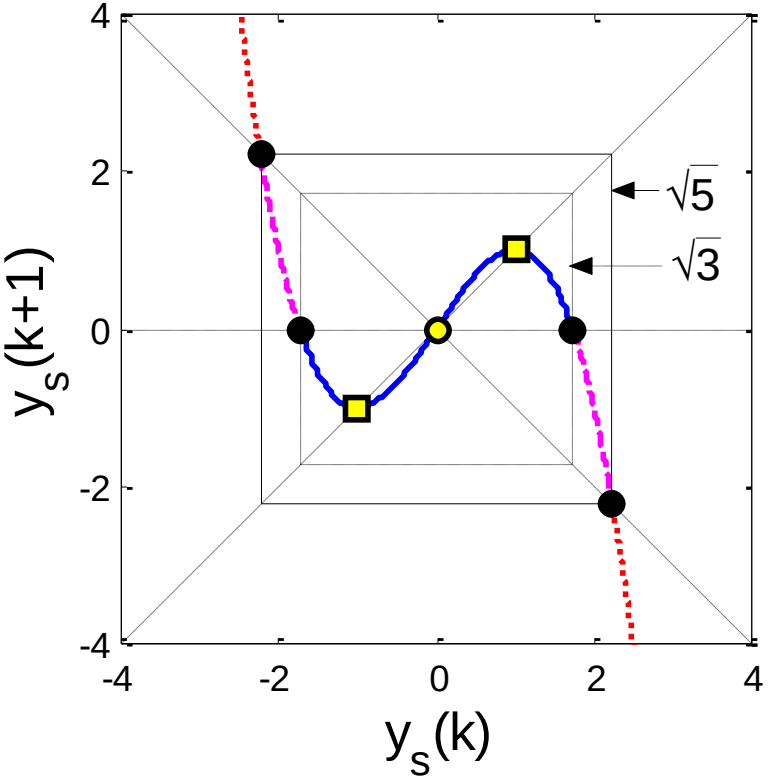
$$y_d(k+1) = y_d(k) \cdot (2 - y_d(k)) \qquad y_d(k) \to \left. 1 \right|_{k \to \infty}$$

$$e_d(k) = y_d(k) - 1$$

$$\boxed{e_d(k+1) = -e_d(k)^2}$$

Initial Condition for 1/sqrt(*N*)?

$$y_s(k+1) = \frac{y_s(k)}{2} \cdot (3 - y_s(k)^2) \qquad y_s(k) \to 1\Big|_{k\to\infty}$$



Convergence: $0 < y_s(0) < \sqrt{3}$

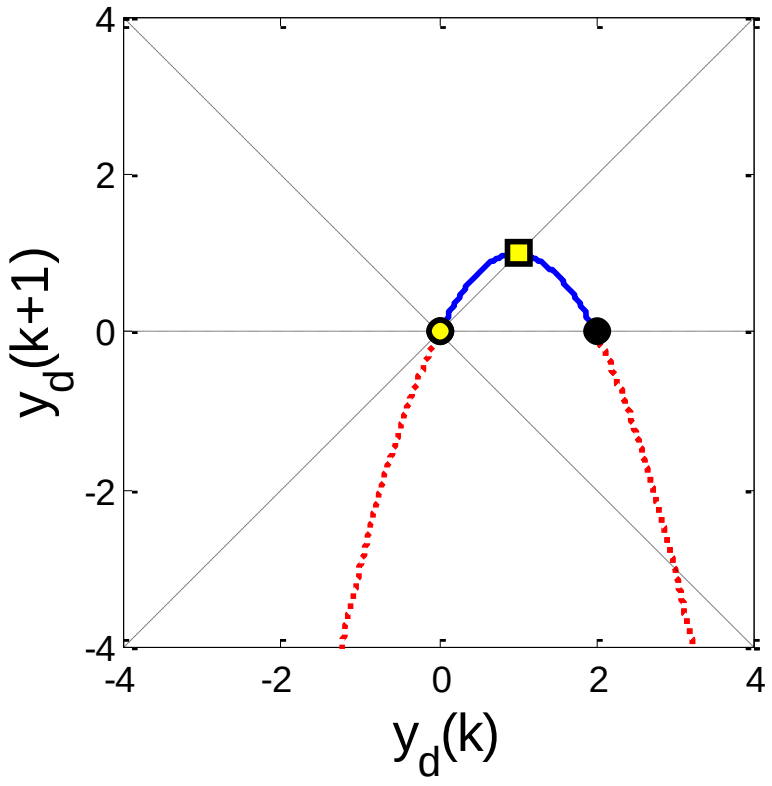
Conv. stripes: $\sqrt{3} < y_s(0) < \sqrt{5}$

Divergence: $y_s(0) > \sqrt{5}$

Initial Condition for 1/*N* ?

$$y_d(k+1) = y_d(k) \cdot (2 - y_d(k))$$

$$y_d(k) \to 1\Big|_{k\to\infty}$$



Convergence: $0 < y_d(0) < 2$

Divergence: otherwise

1/sqrt(*N*): Relative Error

$$x_{n+1} = \frac{x_n}{2} \cdot \left(3 - N \cdot x_n^2\right)$$

$$x_n \to \left.\frac{1}{\sqrt{N}}\right|_{n \to \infty}$$

$$x_n = \frac{y_n}{\sqrt{N}}$$

$$y_{n+1} = \frac{y_n}{2} \cdot \left(3 - y_n^2\right)$$

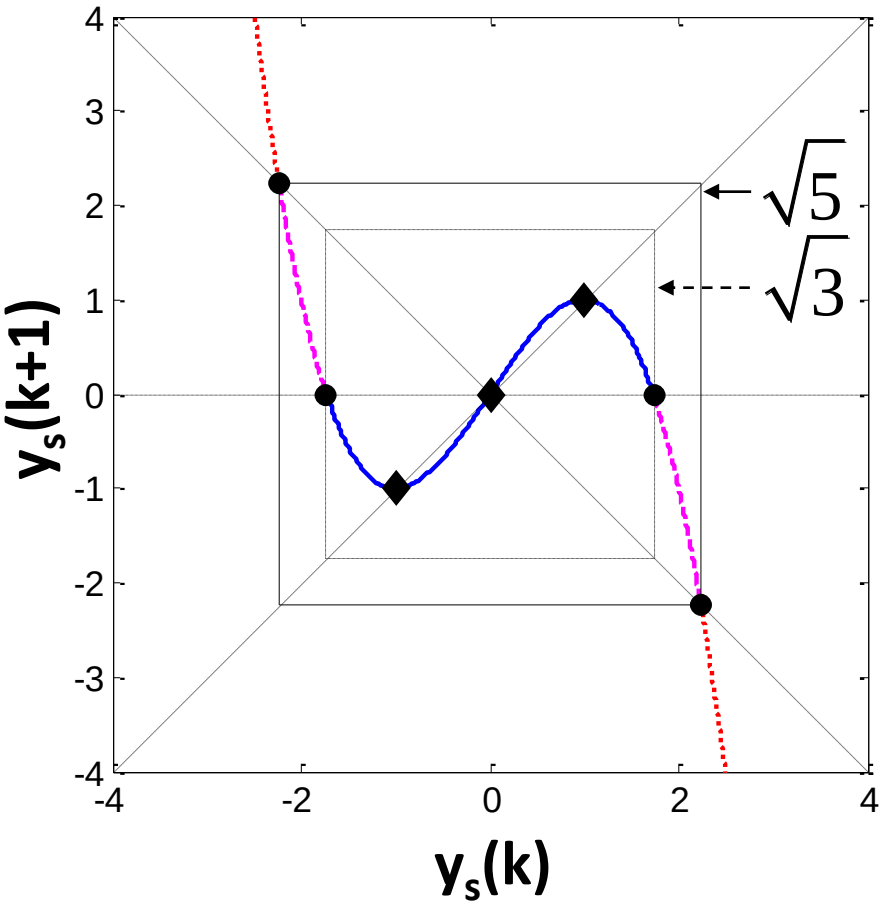
Error: $e_n = 1 - y_n$

$$e_{n+1} = \frac{3}{2} \cdot e_n^2 - \frac{1}{2} \cdot e_n^3$$

1/sqrt(*N*): Convergence Analysis

$$y_{n+1} = \frac{y_n}{2} \cdot \left(3 - y_n^2\right)$$

$$y_n \to 1\Big|_{n\to\infty}$$

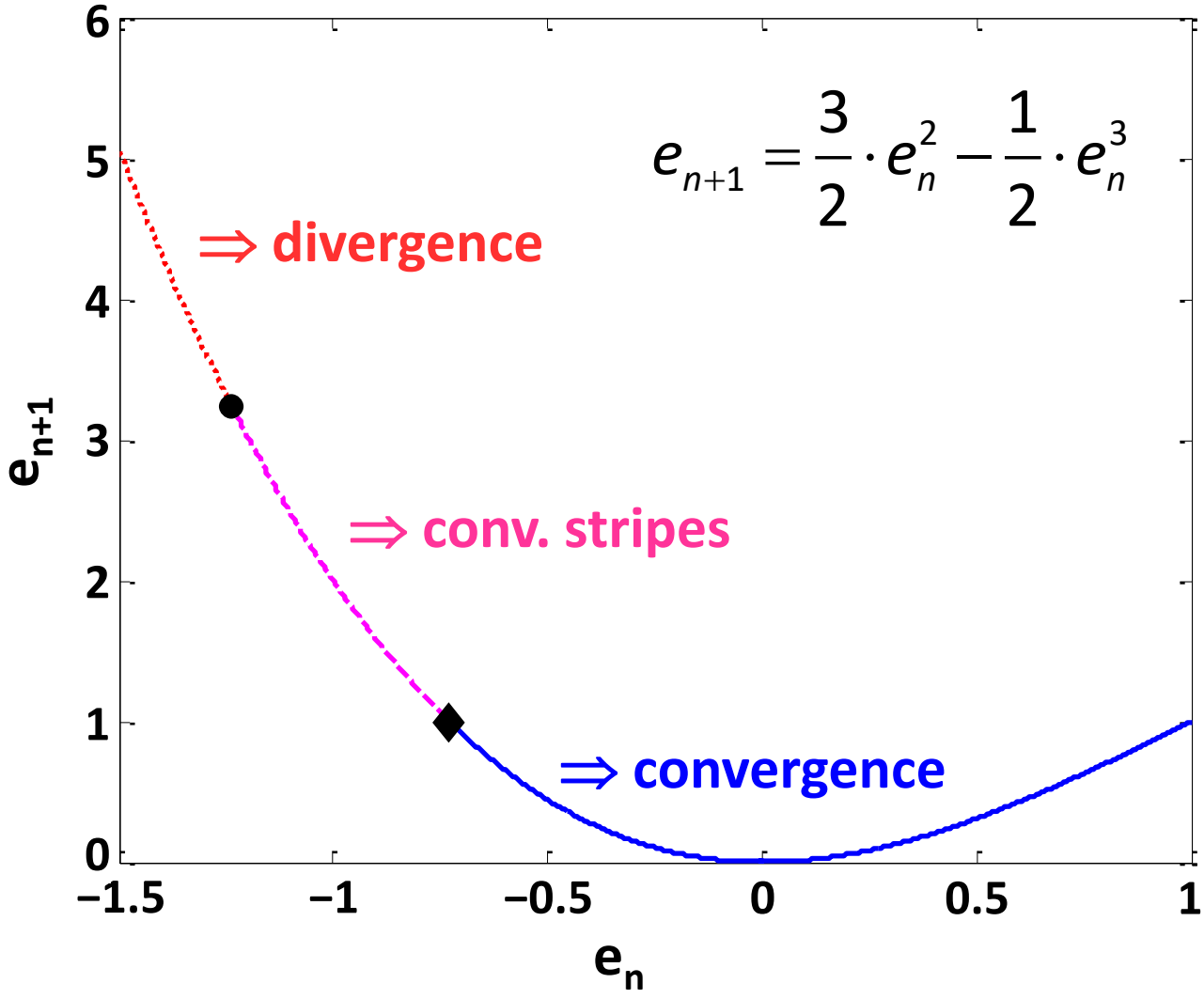


$0 < y_0 < \sqrt{3}$ ⇒ **convergence**

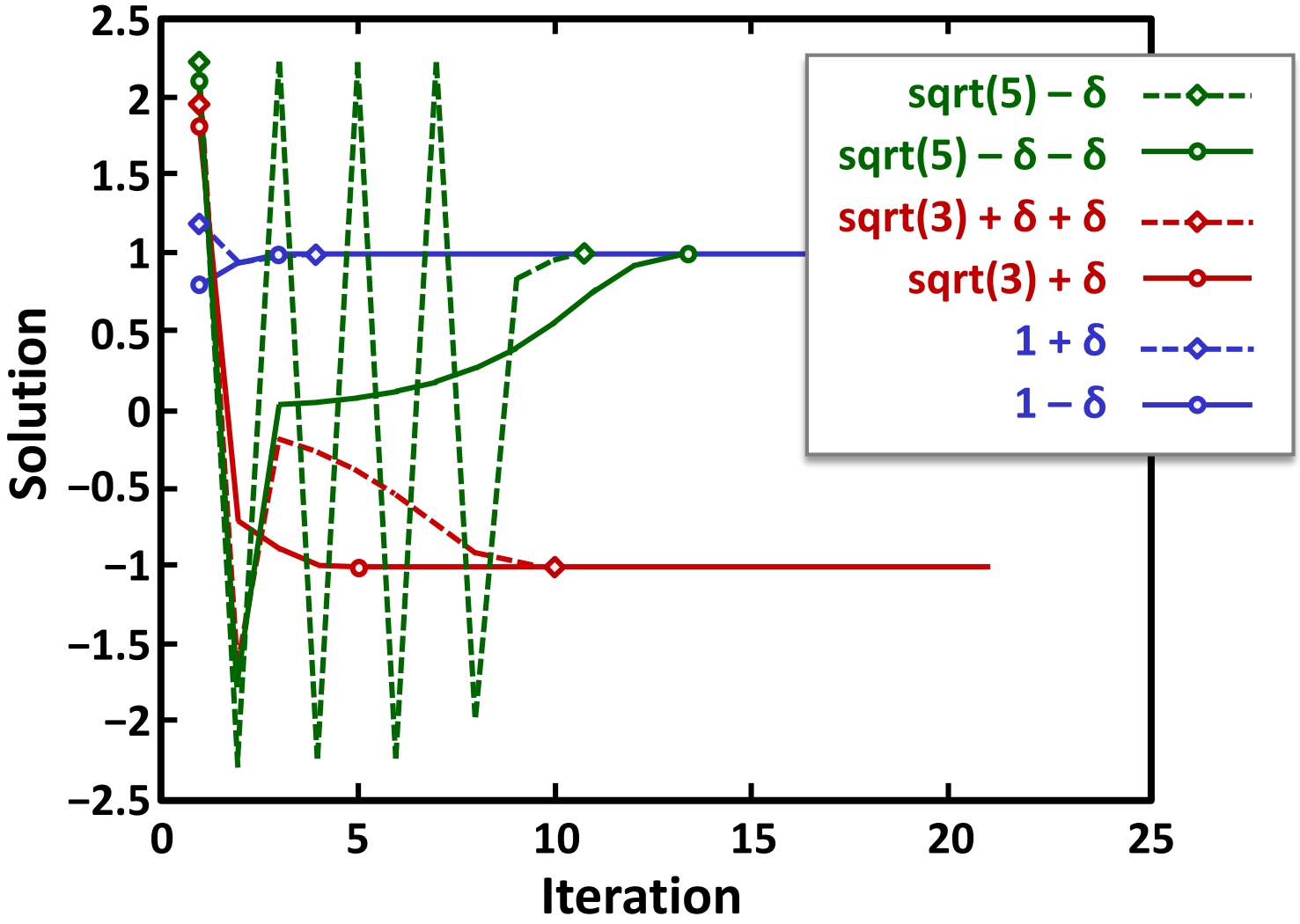
$\sqrt{3} < y_0 < \sqrt{5}$ ⇒ **conv. stripes**

$y_0 > \sqrt{5}$ ⇒ **divergence**

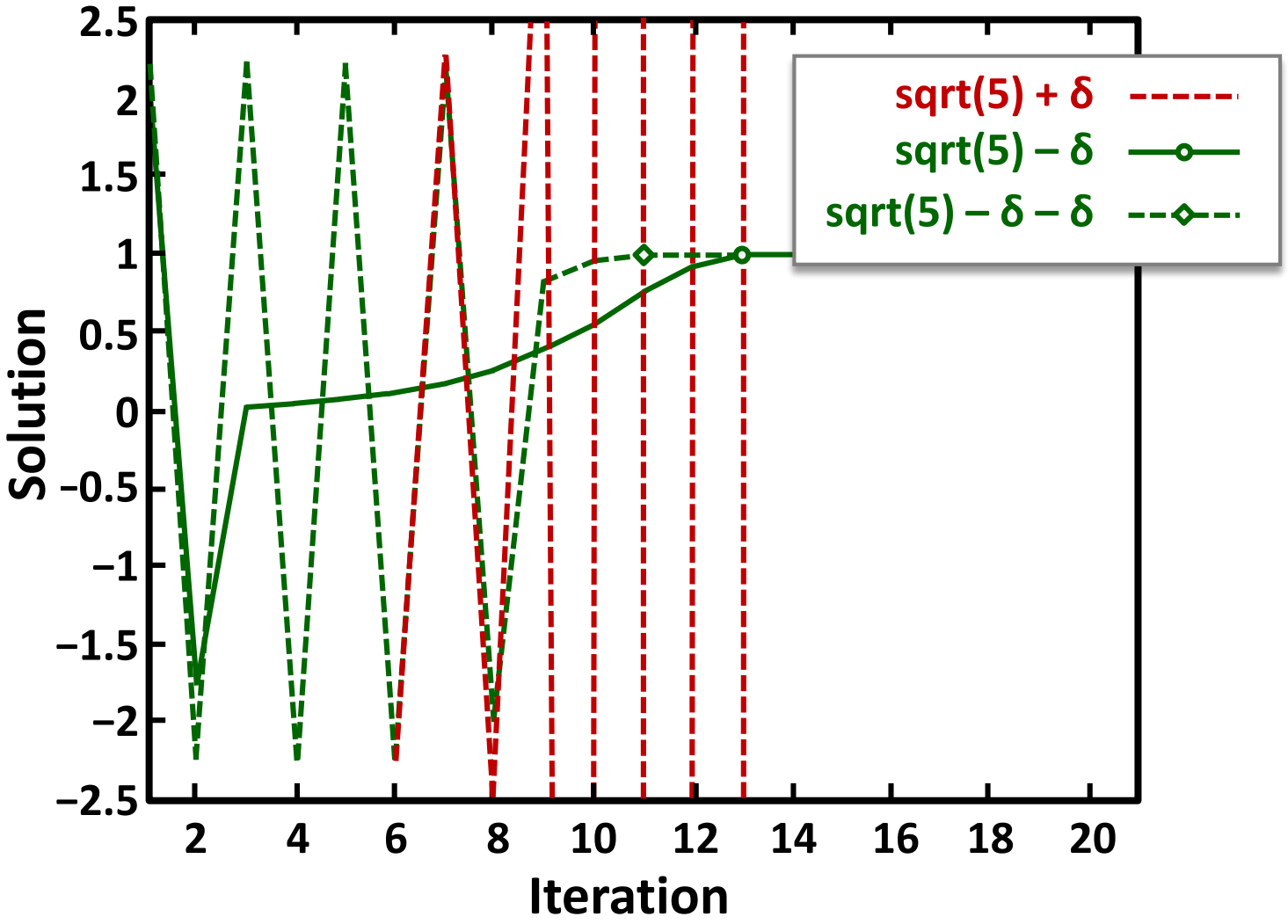
1/sqrt(*N*): Error Function



Example: Choosing the Initial Condition



Initial Condition > sqrt(5) : Divergence



1/sqrt(): Picking Initial Condition

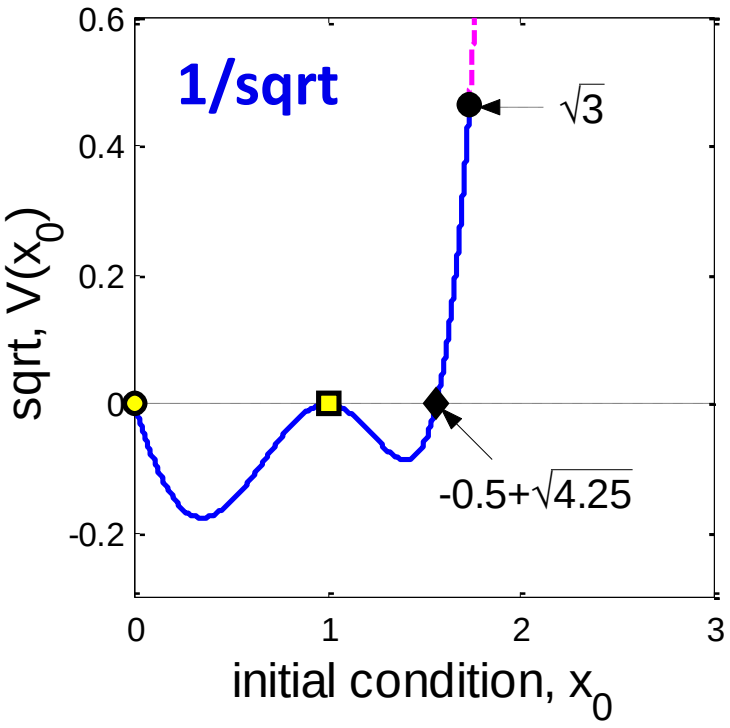
$$x_{n+1} = \frac{x_n}{2} \cdot \left(3 - N \cdot x_n^2\right) \qquad x_n \to \left.\frac{1}{\sqrt{N}}\right|_{n\to\infty}$$

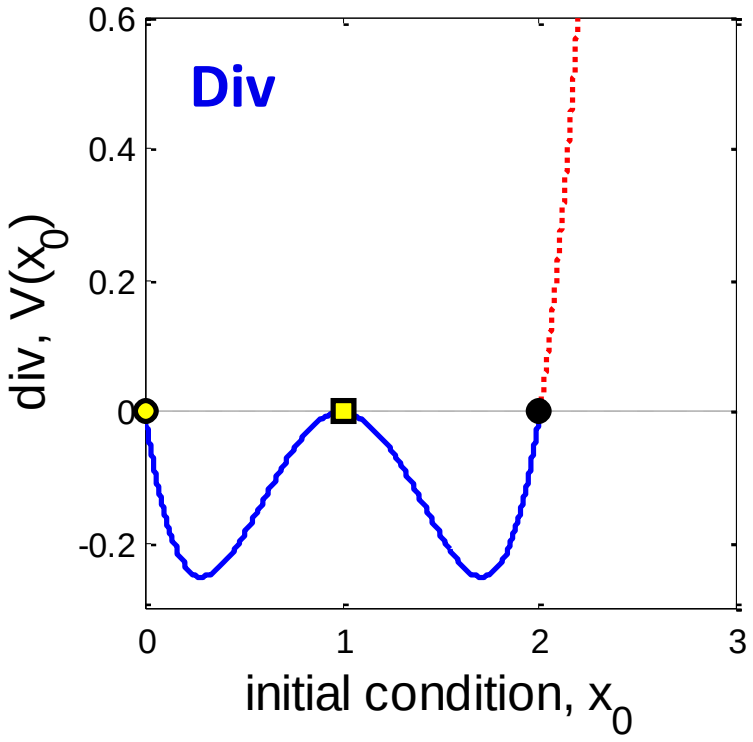
- **Equilibriums** $0, \quad \pm\frac{1}{\sqrt{N}}$
- **Initial condition**
  - Take: $V(x_n) = \left(x_n - \frac{1}{\sqrt{N}}\right)^2$ **(8.1)**
  - Find x_0 such that:

    $V(x_{n+1}) - V(x_n) < 0 \quad \forall n \quad (n = 0,1,2,\ldots)$ **(8.2)**
  - Solution: $S = \{x_0 : V(x_0) < a\}$ **(8.3)**

"Level set" $V(x_0) = a$ is a convergence bound
→ Local convergence (3 equilibriums)

Descending Absolute Error





$$V_s(x_0) = \frac{x_0}{4} \cdot (x_0 - 1)^2 \cdot (x_0 + 1) \cdot (x_0^2 + x_0 - 4)$$

$$V_d(x_0) = x_0 \cdot (x_0 - 1)^2 \cdot (x_0 - 2)$$

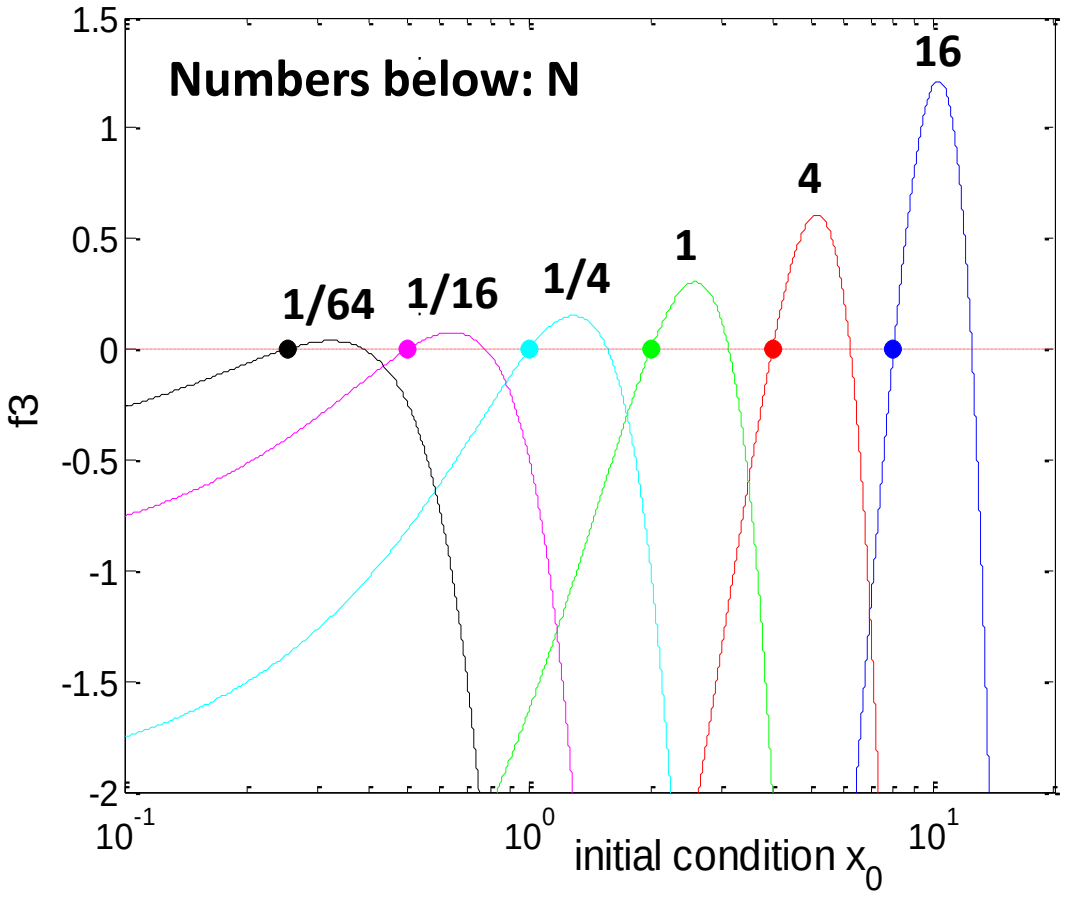
Descending error:

$$E(x_k) = (x_k - 1)^2 \qquad V(x_k) = E(x_{k+1}) - E(x_k) < 0 \qquad \forall k$$

1/sqrt(*N*): Picking Initial Condition (Cont.)

(8.2) →

$$\frac{x_0}{2}\cdot(1-N\cdot x_0^2)\cdot\underbrace{\left(\frac{5}{2}x_0-\frac{N}{2}x_0^3-\frac{2}{\sqrt{N}}\right)}<0$$



Roots:

$$x_1=\frac{1}{\sqrt{N}}$$

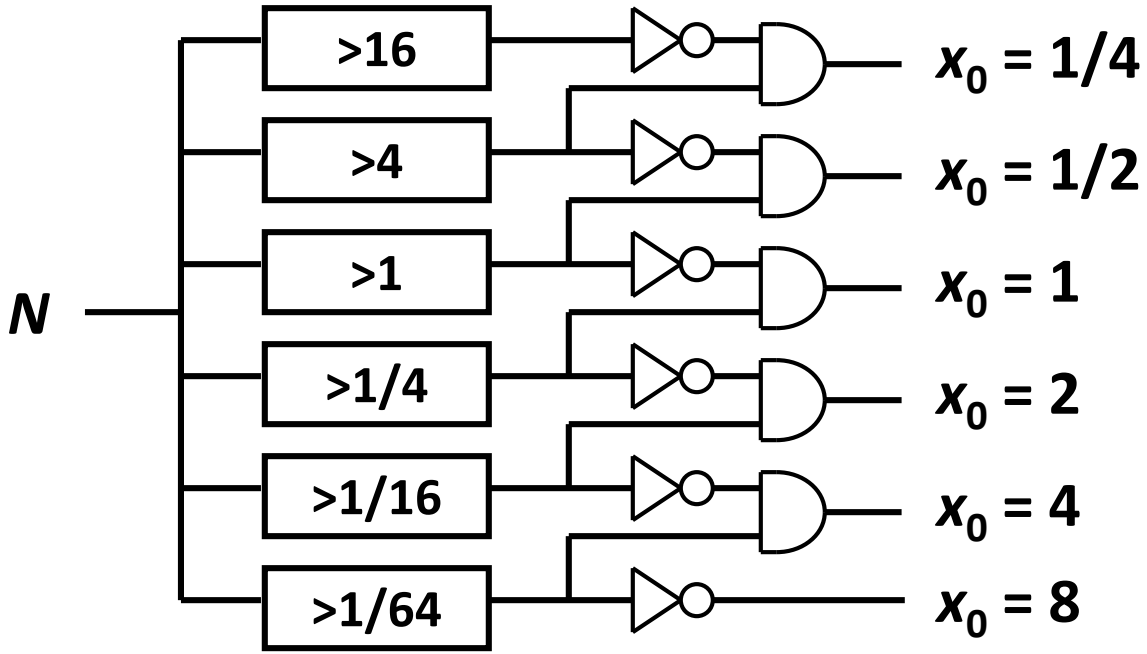
$$x_{2,3}=\frac{1}{2\cdot\sqrt{N}}\left(-1\pm\sqrt{17}\right)$$

Max:

$$x_{0M}=\sqrt{\frac{5}{3N}}$$

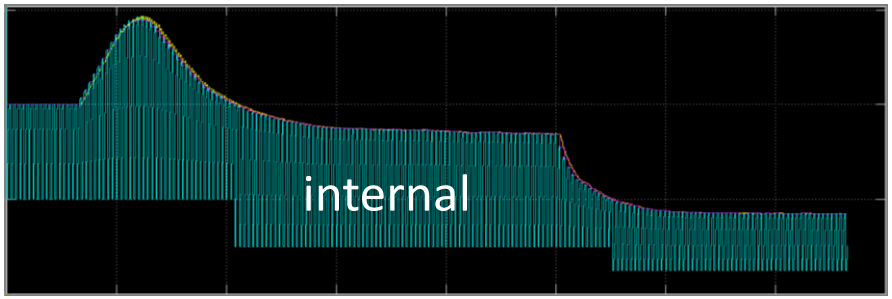
Initial Condition Circuit

(8.3) → $$\frac{1}{\sqrt{N}} - \sqrt{a} < x_0 < \frac{1}{\sqrt{N}} + \sqrt{a}$$

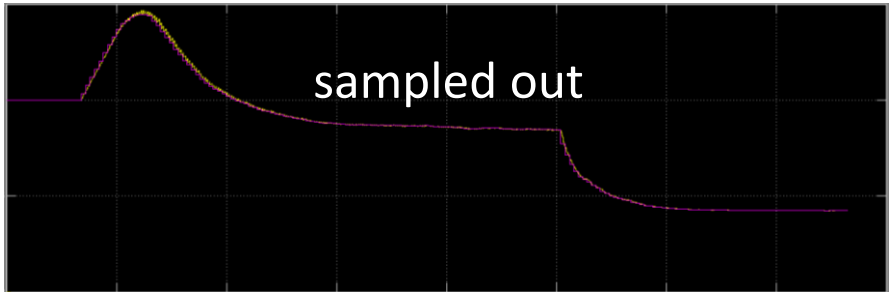


Left: Internal Node | Right: Sampled Output

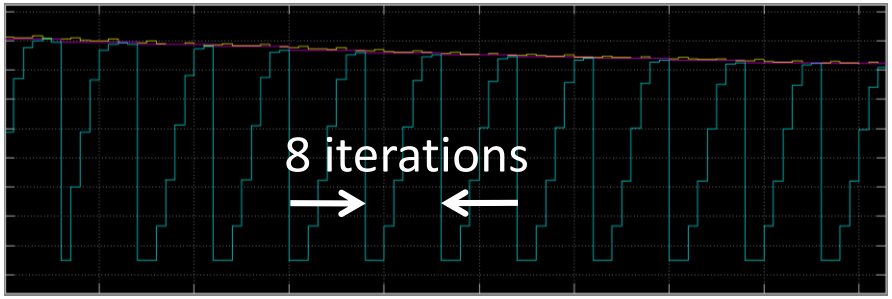
Internal node and output



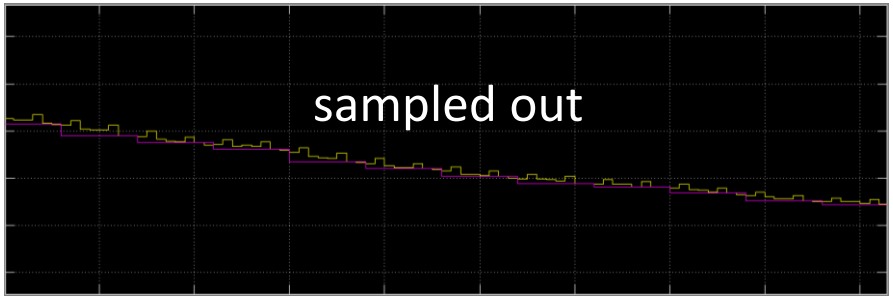
Sampled output of the left plot



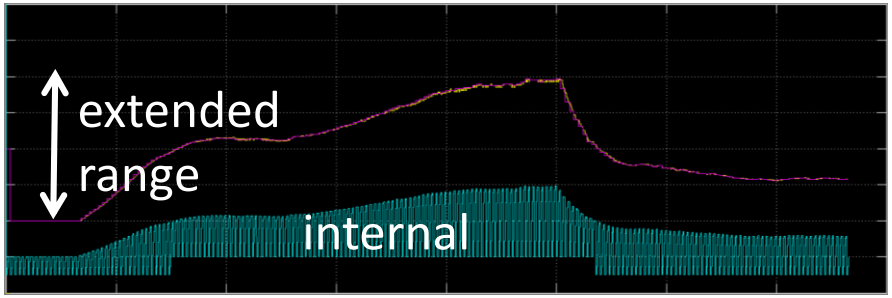
Zoom in: convergence in 8 iterations



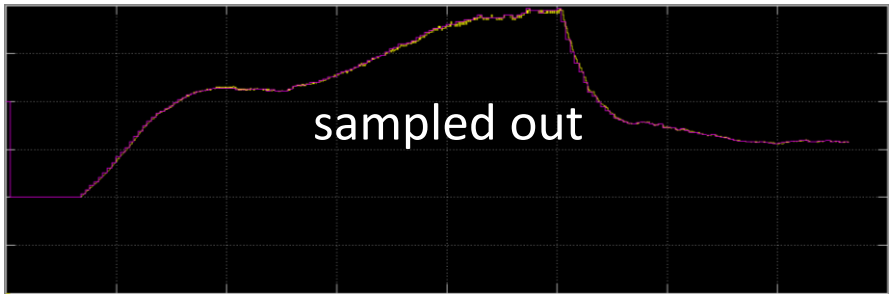
Sampled output of the left plot



Internal divide by 2 extends range



Sampled output of the left plot

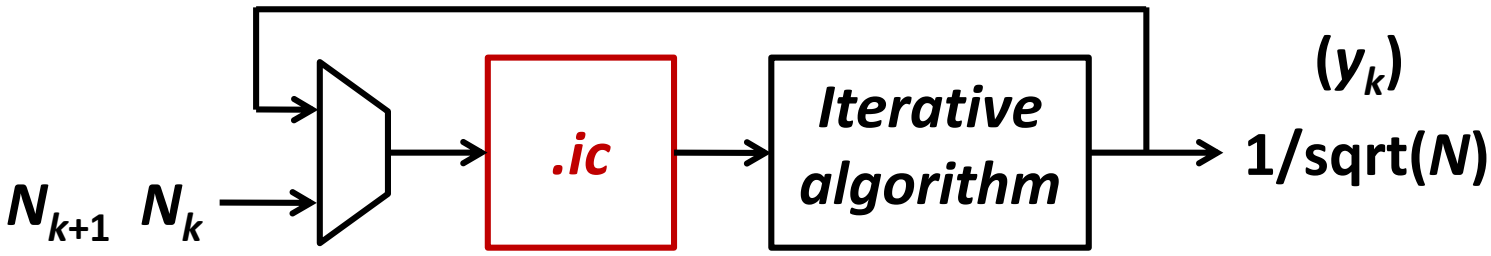


Convergence Speed

- **# iterations required for specified accuracy**

Target relative error (%)	0.1%	1%	5%	10%
e_0: 50%, # iter (sqrt/div)	5 / 4	5 / 3	4 / 3	3 / 2
e_0: 25%, # iter (sqrt/div)	3 / 3	3 / 2	2 / 2	2 / 1

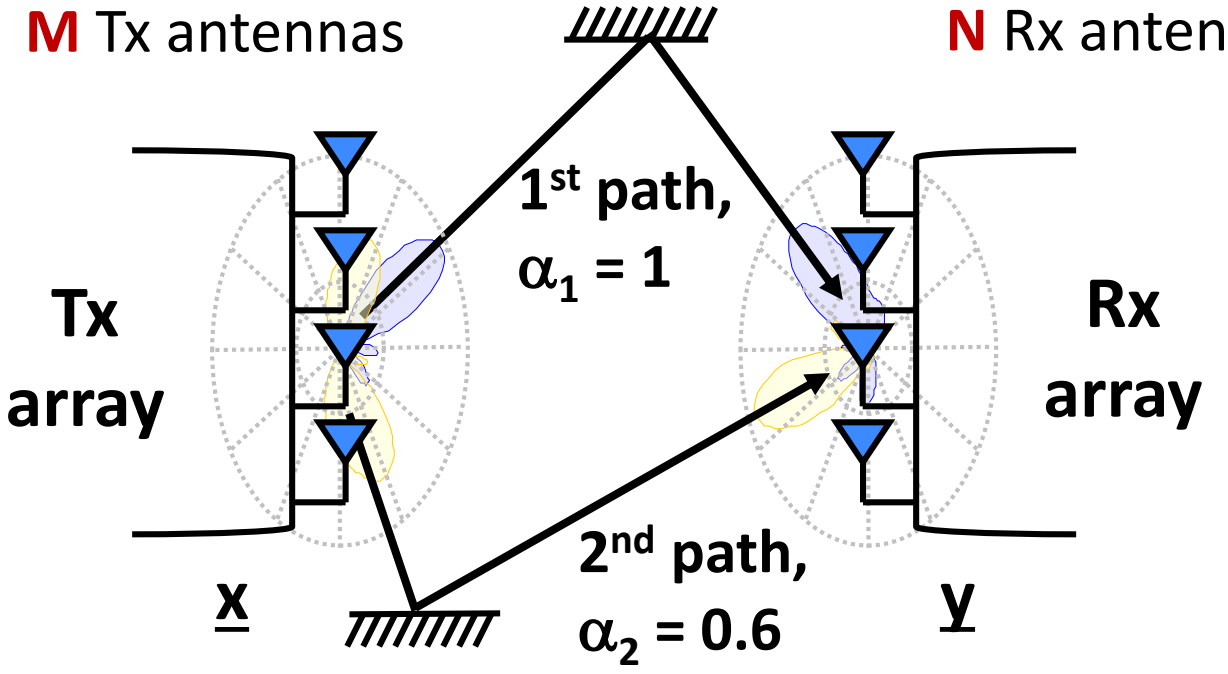
- **Adaptive algorithm**
  - current result → .ic for next iteration



Hierarchical Extension:
Blind Tracking SVD

Multi-Path MIMO Channel

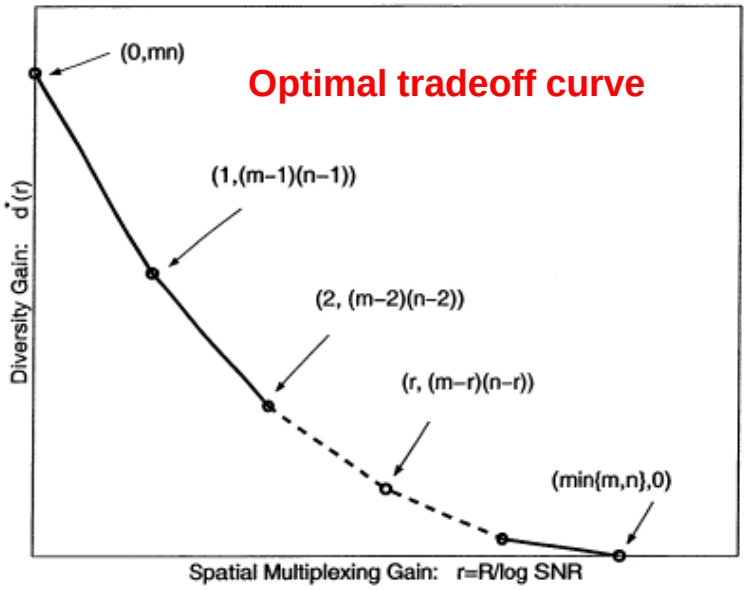
- **Multi-path averaging can be used to improve robustness or increase capacity of a wireless link**



MIMO channel: Matrix H_{MxN}

MIMO Diversity-Multiplexing Tradeoff

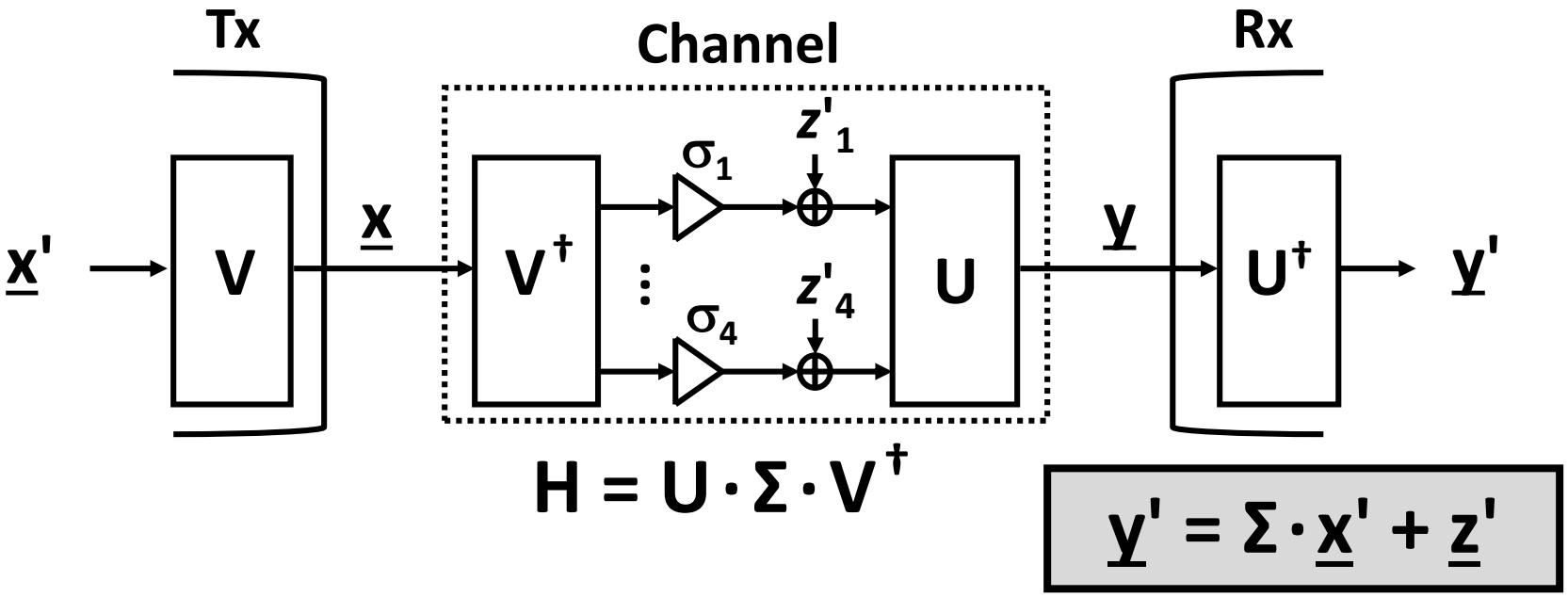
- **Sphere decoding can extract both diversity and spatial multiplexing gains**
  - Diversity gain ***d*** : error probability decays as $1/SNR^{d}$
  - Multiplexing gain ***r*** : channel capacity increases ~ ***r***·log (SNR)



SVD: spatial mux gain

L. Zheng and D. Tse, "Diversity and Multiplexing: A Fundamental Tradeoff in Multiple-Antenna Channels," *IEEE Trans. Inf. Theory*, vol. 49, no. 5, pp. 1073-1096, May 2003.

SVD Channel Decoupling

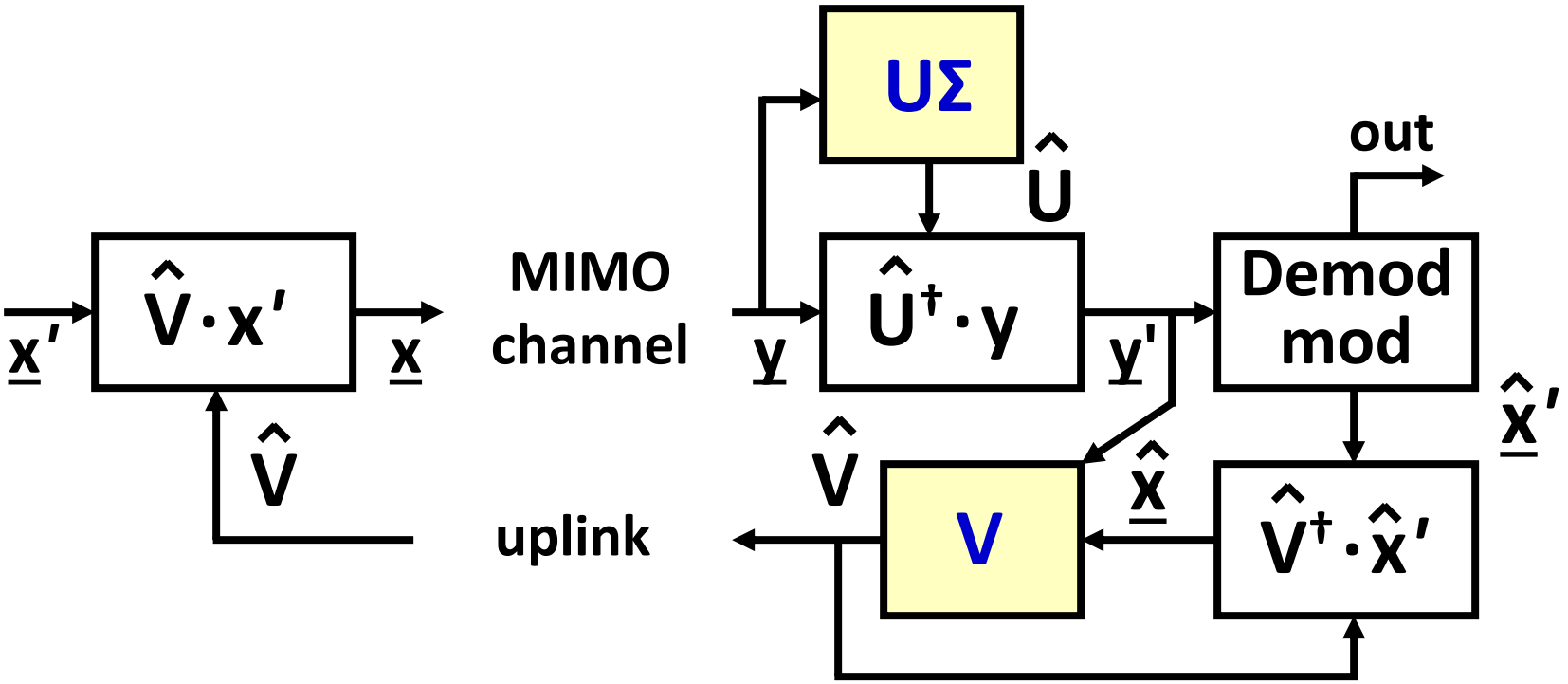


Complexity: 100's of add, mult; also div, sqrt

Architecture that minimizes power and area?

A. Poon, D. Tse, and R.W. Brodersen, "An Adaptive Multiple-Antenna Transceiver for Slowly Flat-Fading Channels," *IEEE Trans. Communications,* vol. 51, no. 13, pp. 1820-1827, Nov. 2003.

Adaptive Blind-Tracking SVD



- **MIMO decoder specifications**
  - 4x4 antenna system; variable PSK modulation
  - 16 MHz channel bandwidth; 16 sub-carriers

LMS-Based Estimation of UΣ

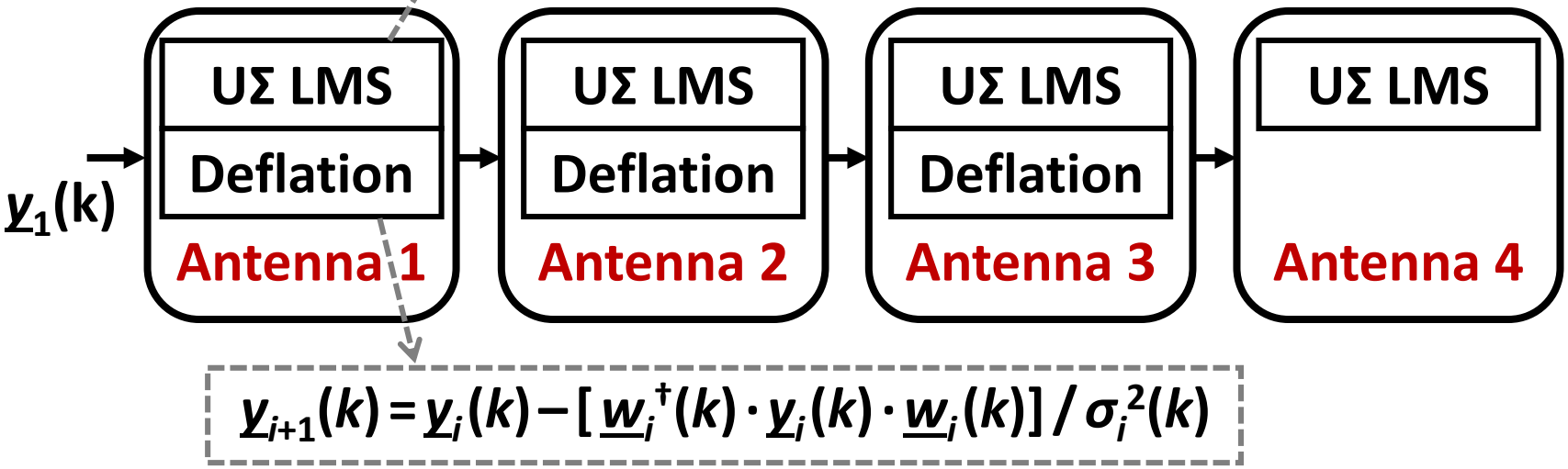
- **This complexity is hard to optimize in RTL**
  - 270 adders, 370 multipliers, 8 sqrt, 8 div

$$\underline{w}_i(k) = \underline{w}_i(k-1) + \mu_i \cdot [\underline{y}_i(k) \cdot \underline{y}_i^{\dagger}(k) \cdot \underline{w}_i(k-1) - \sigma_i^2(k-1) \cdot \underline{w}_i(k-1)]$$

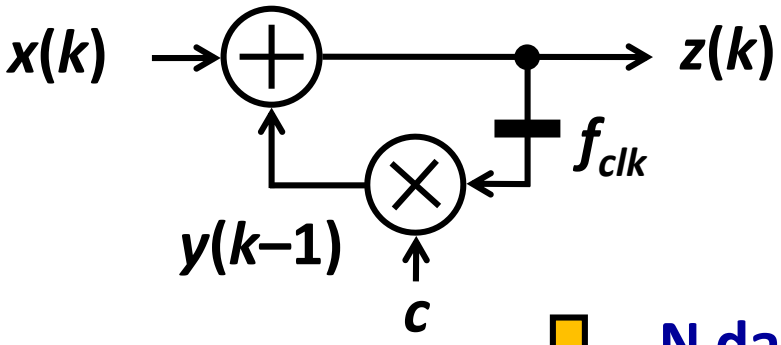
$$\sigma_i^2(\mathrm{k}) = \underline{w}_i^{\dagger}(k) \cdot \underline{w}_i(k)$$

$$\underline{u}_i(k) = \underline{w}_i(k) / \sqrt{\sigma_i^2(k)}$$

$$(i = 1, 2, 3, 4)$$



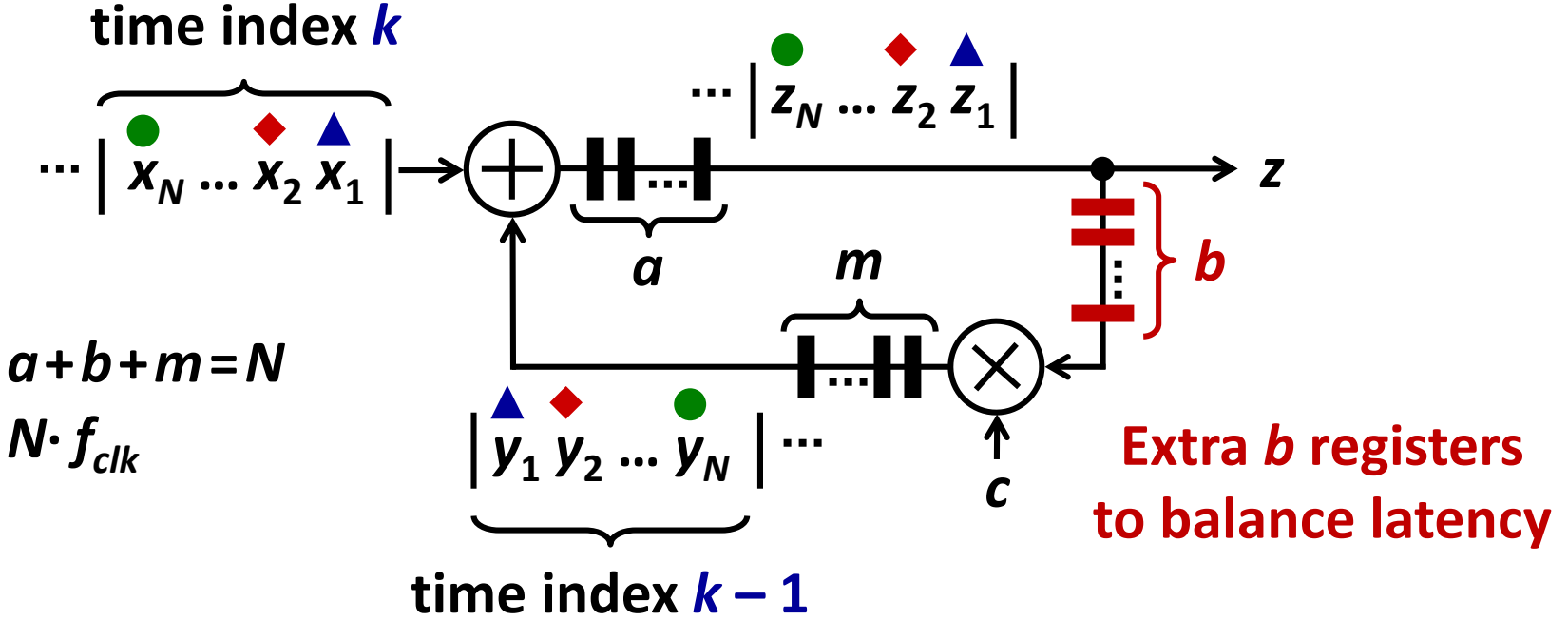
Multi-Carrier Data-Stream Interleaving



Recursive operation:

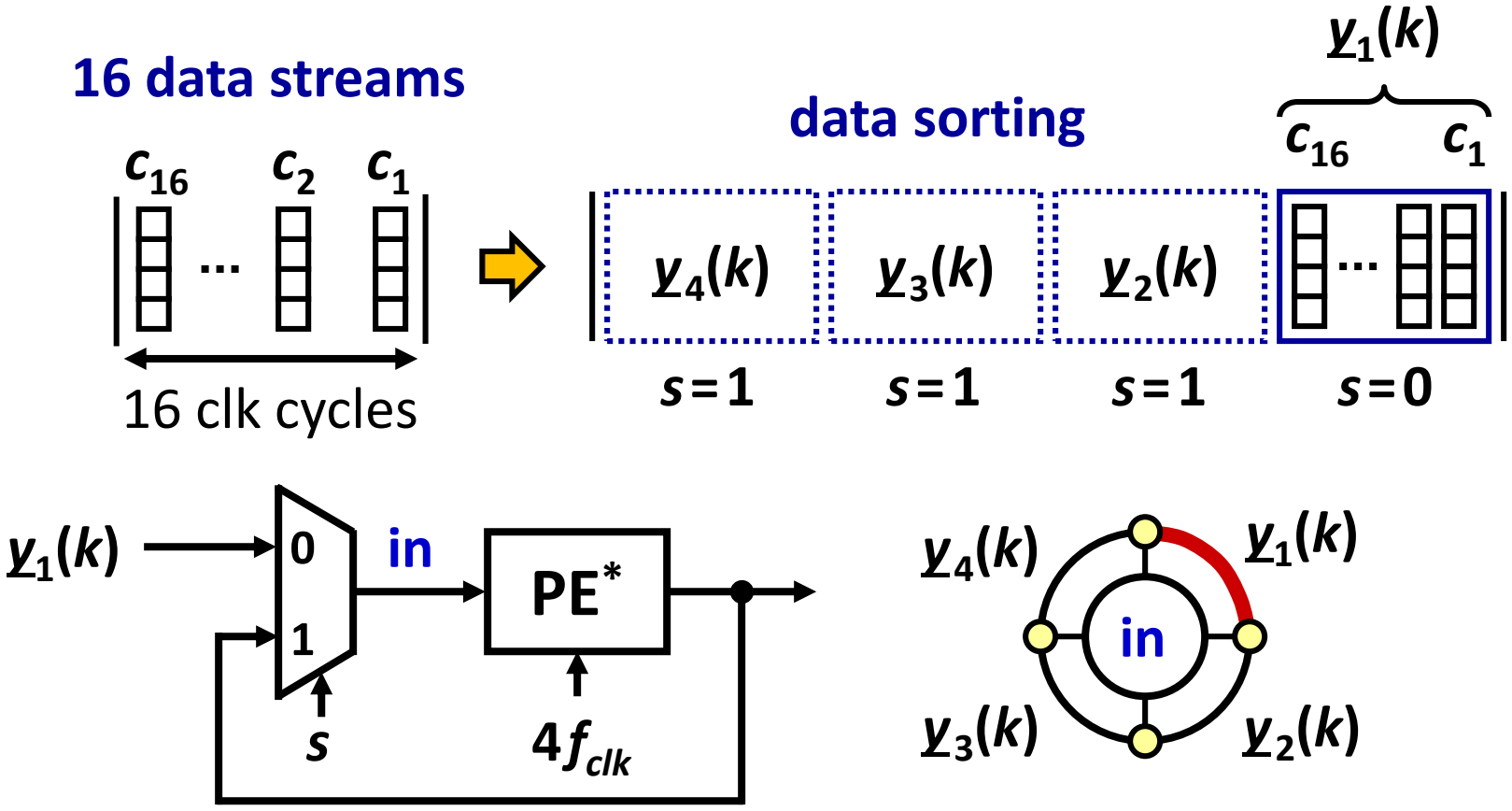
$z(k) = x(k) + c \cdot z(k-1)$

N data streams: $x_1, x_2, \ldots, x_N$



Extra *b* registers to balance latency

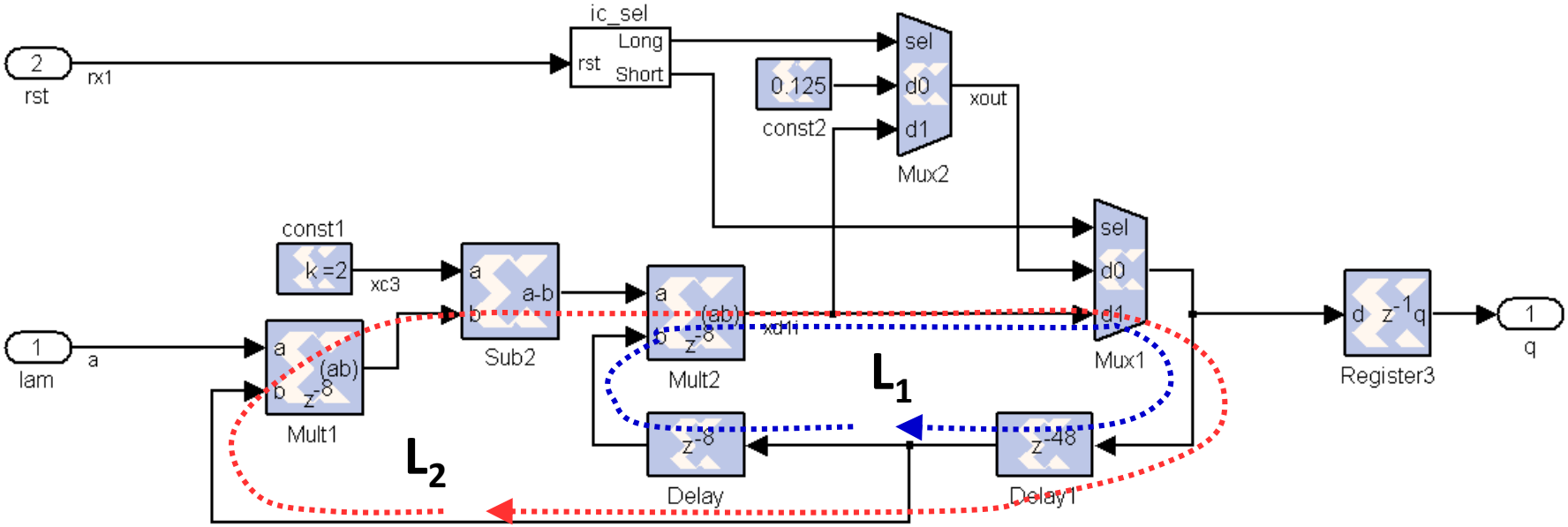
Architecture Folding



- **Folding = up-sampling & pipelining**
  - Reduced area (shared datapath logic)

Challenge: Loop Retiming

- **Iterative division:** $y_d(k+1) = y_d(k)\cdot(2 - y_d(k))$



- **Loop constraints:**
  - L_1: $m + u + d_1 = N$
  - L_2: $2m + a + u + d_2 = N$

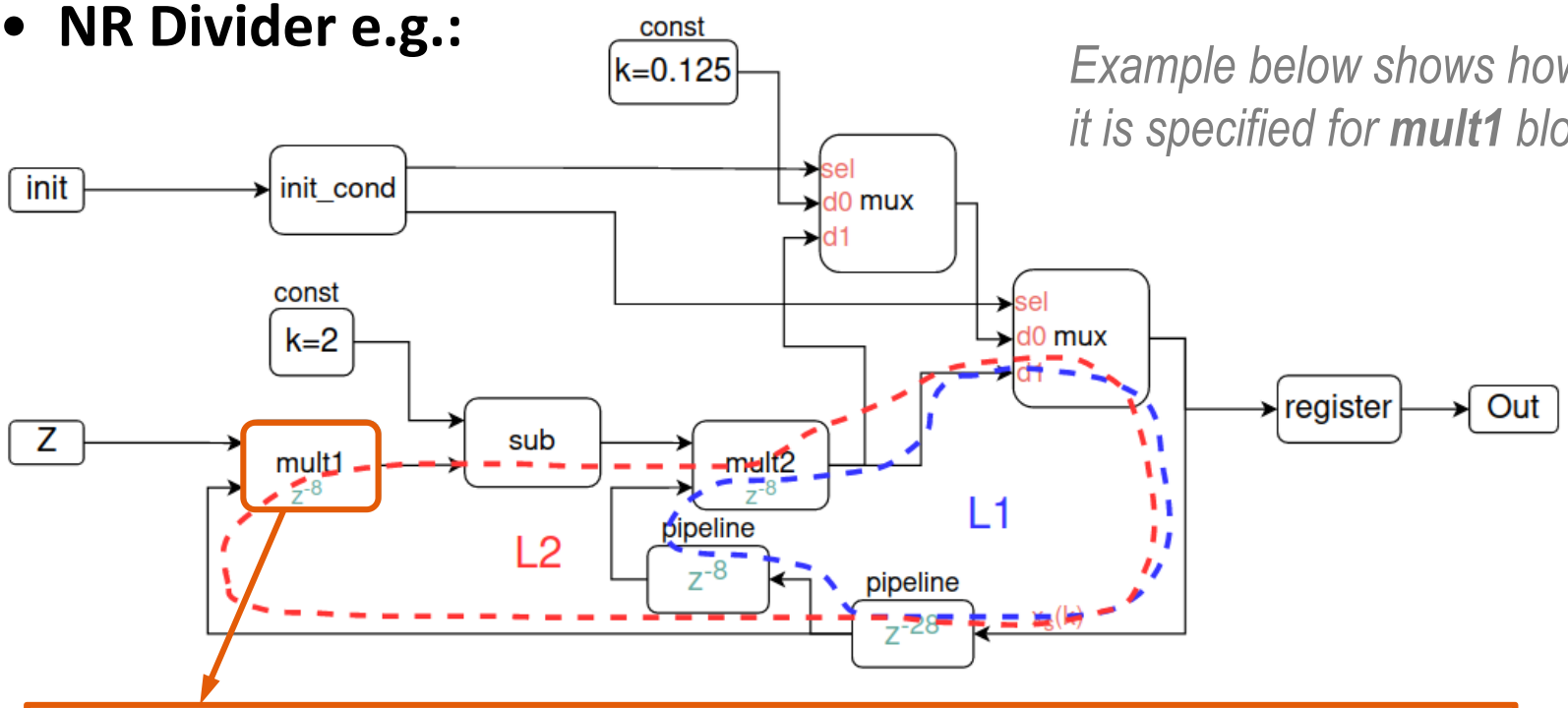
Opt: m, a, u d_1, d_2

- **Latency parameters (m, a) are a function of cycle time**

Quantization and Overflow in PyGears

Operators in loops need saturation & rounding

- **NR Divider e.g.:**



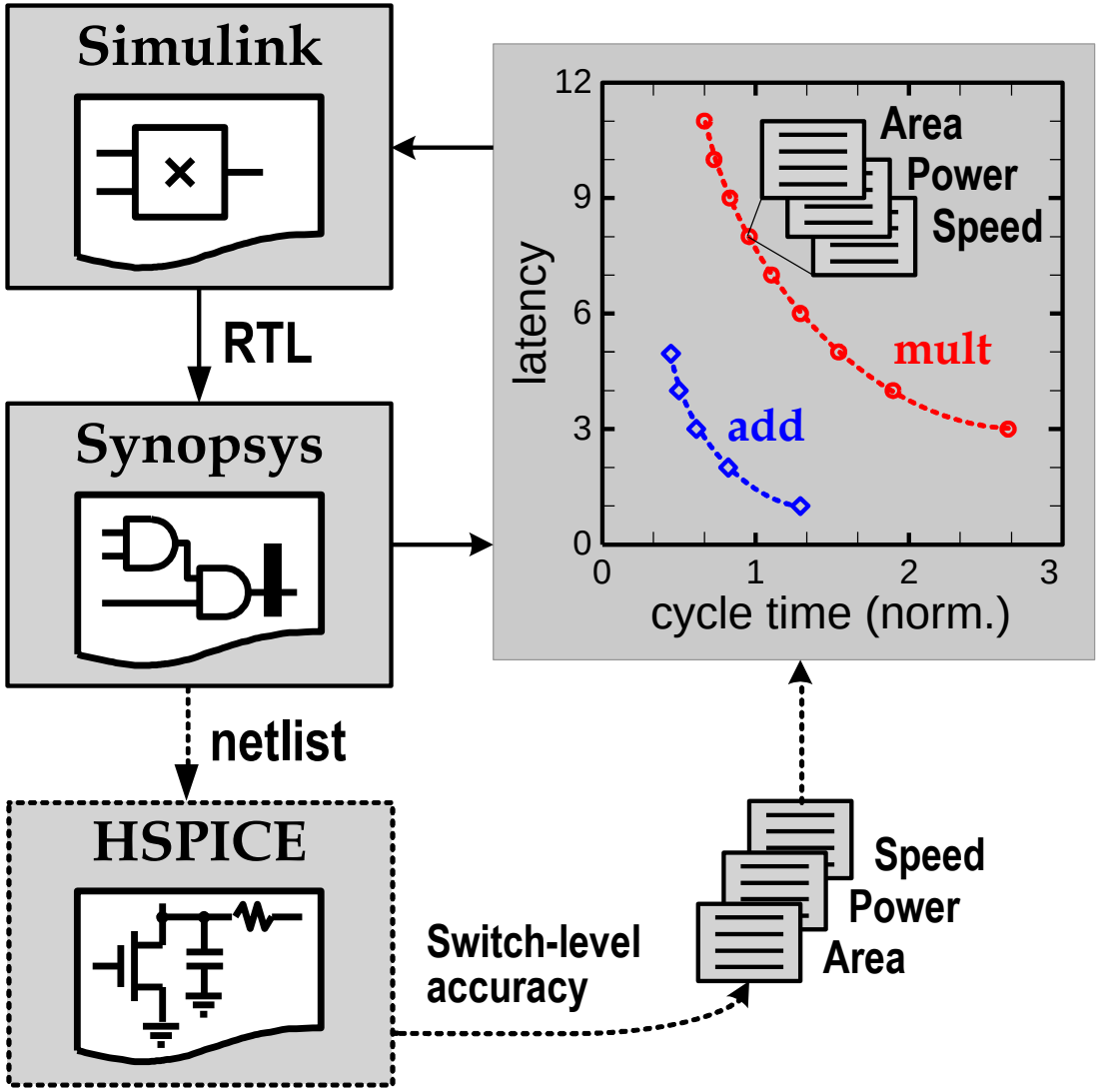
*Example below shows how it is specified for **mult1** block*

```
m1 = mult_dsp(z, x_d,
              t=Fixp[6, 14],
              overflow=Overflow.SATURATE,
              quantization=Quantization.ROUND,
              latency=8)
```

output format

latency

Cycle Time → Block Latency

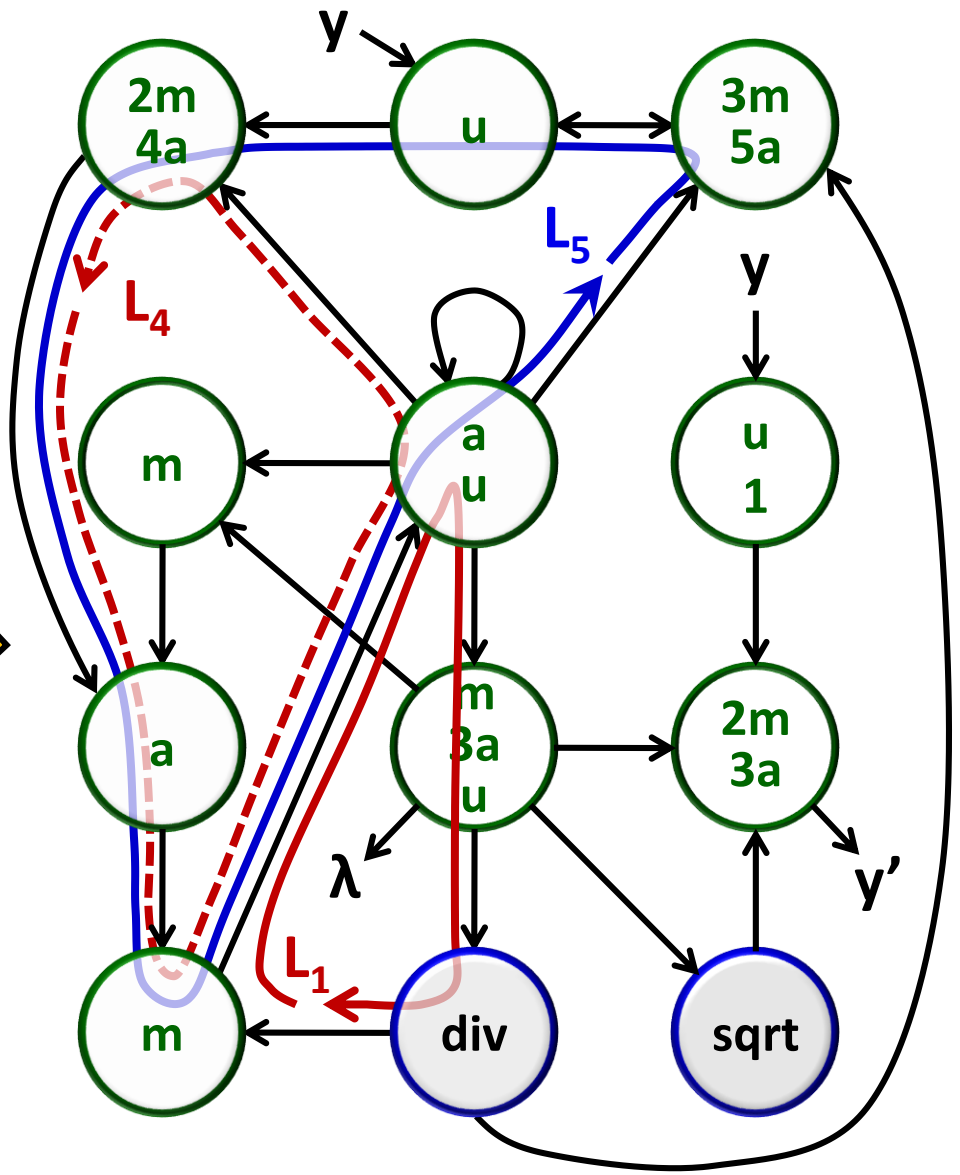


Hierarchical Loop Retiming

- **Divider**
  - IO latency
    - $2m + a + u + 1$ **(div)**
  - Internal Loops
    - $L_1^{(1)}$: $2m + a + u + d_1 = N$
    - $L_2^{(1)}$: $m + u + d_2 = N$

- **Additional constraints (next layer of hierarchy)**
  - $L_1^{(2)}$: ***div*** $+ 2m + 4a + 2u + d_1 = N$
  - . . .
  - $L_4^{(2)}$: $3m + 6a + u + d_4 = N$
  - $L_5^{(2)}$: $6m + 11a + 2u + d_5 = N$ ***+ N***

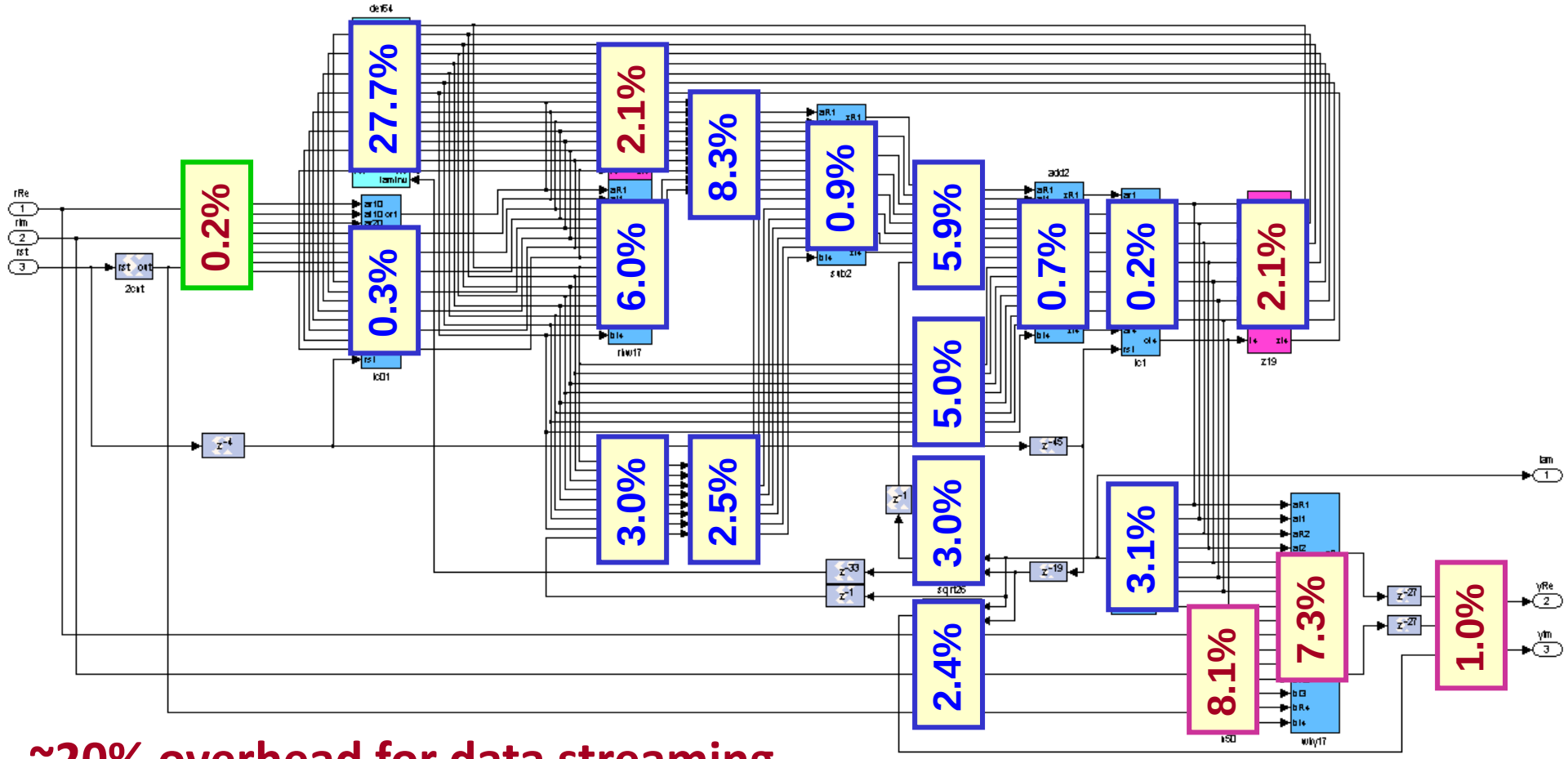
(delayed LMS)



UΣ Block: Power Breakdown

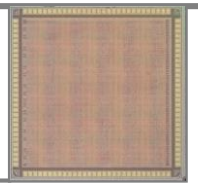
“report_power –hier –hier_level 2”

(one hier level shown here)

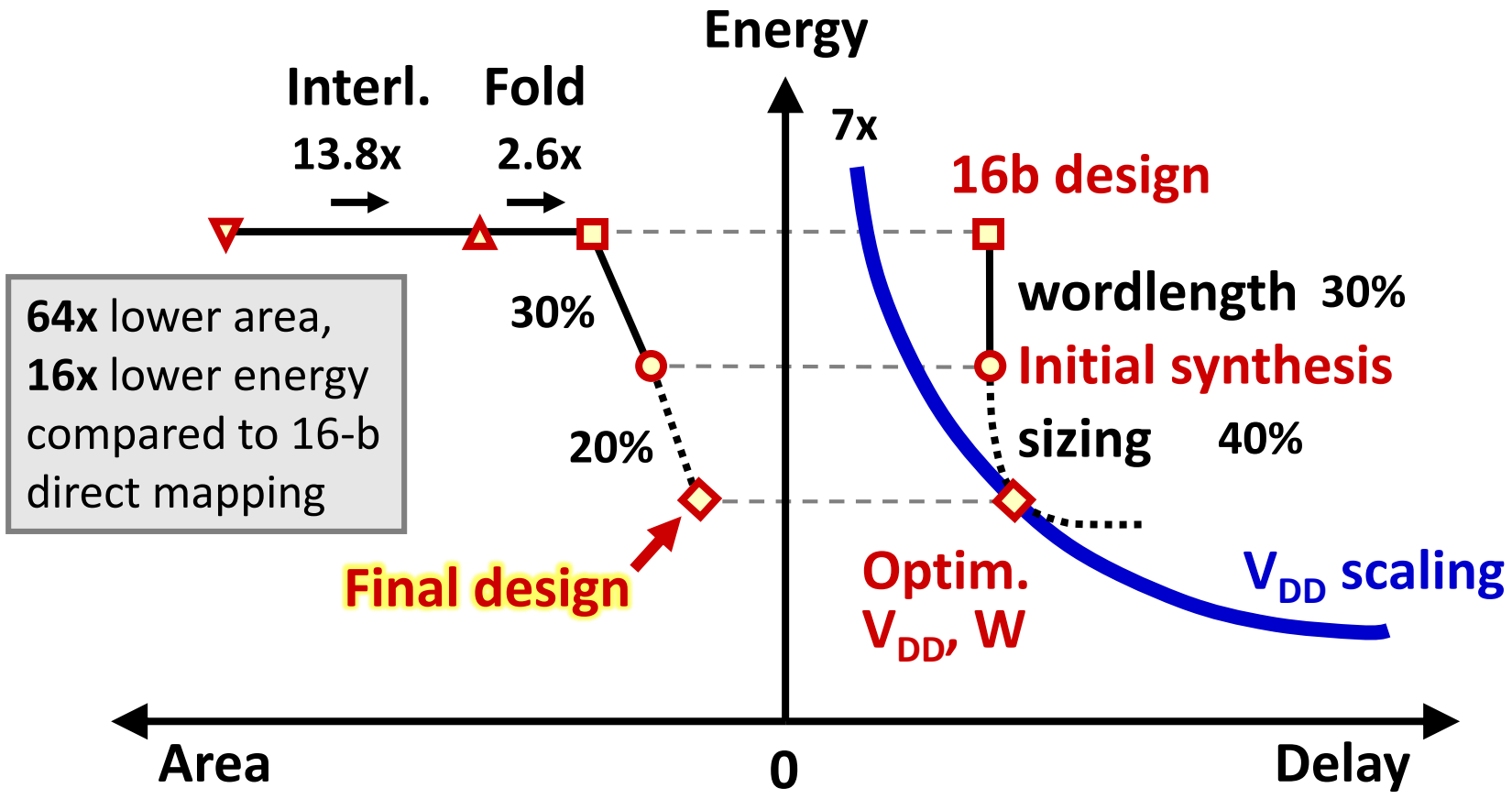


~20% overhead for data streaming

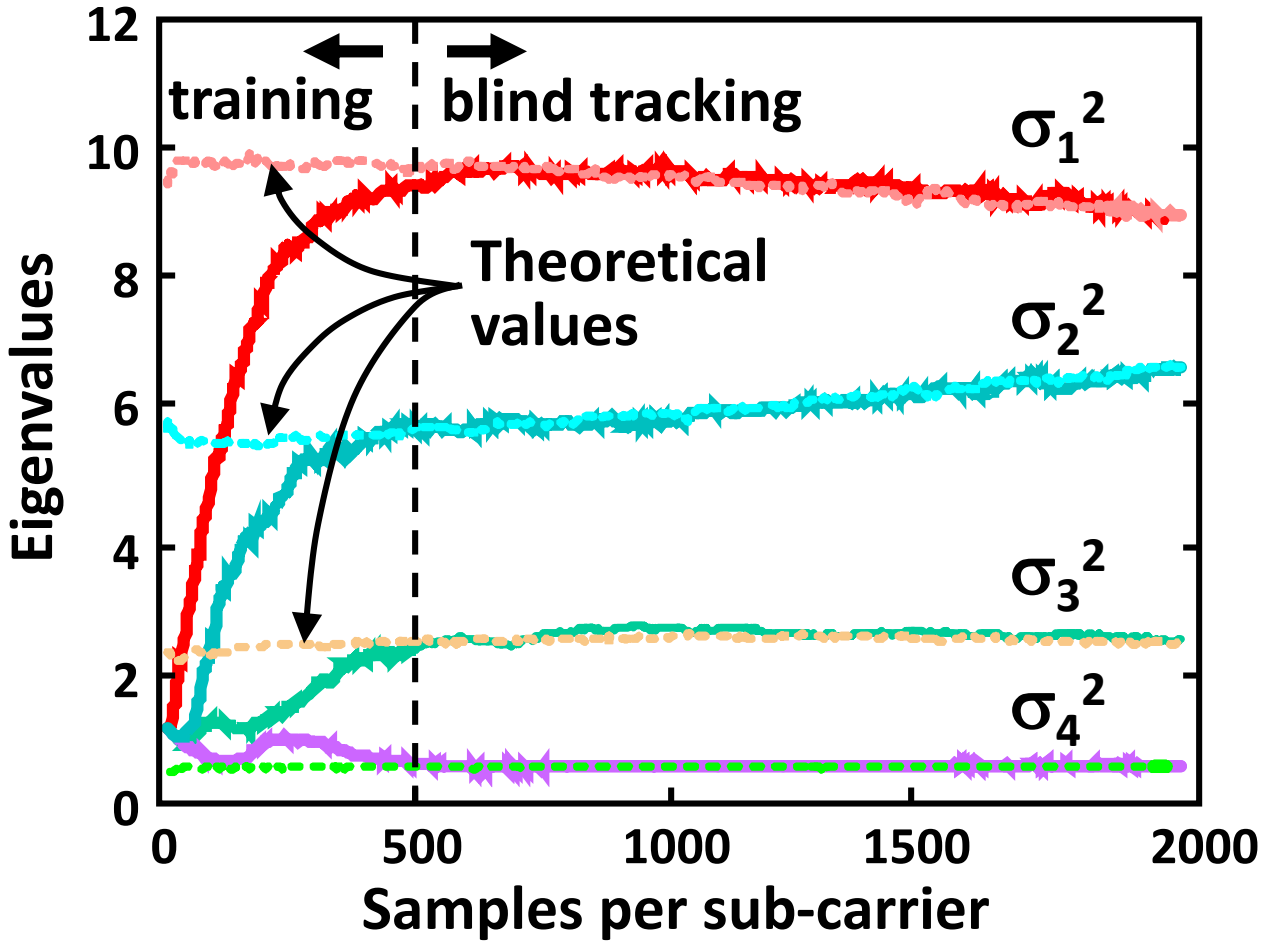
Energy/Area Optimization

90 nm CMOS OP = 12-bit +	**Energy efficiency: 2.1 GOPS/mW** **Area efficiency: 20 GOPS/mm²**	

D. Marković, et al., VLSI'06, pp. 196-197.



Measured Functionality



Summary

- **CORDIC uses angular rotations to compute various functions**
  - One bit of resolution / iteration
- **Netwon-Raphson algorithms for square rooting and division converge faster than CORDIC**
  - Two bits of resolution / iteration
- **Convergence speed greatly depends on the initial condition**
  - The choice of initial condition can be made as to guarantee decreasing absolute error in each iteration
  - For slowly varying inputs, adaptive algorithms can use the result of current iteration as the initial condition
  - Hardware latency depends on the initial condition & accuracy
- **High-level retiming is critical for complex recursive algorithms**