ECE4740: Digital VLSI Design

Lecture 25: Multiplier & CORDIC

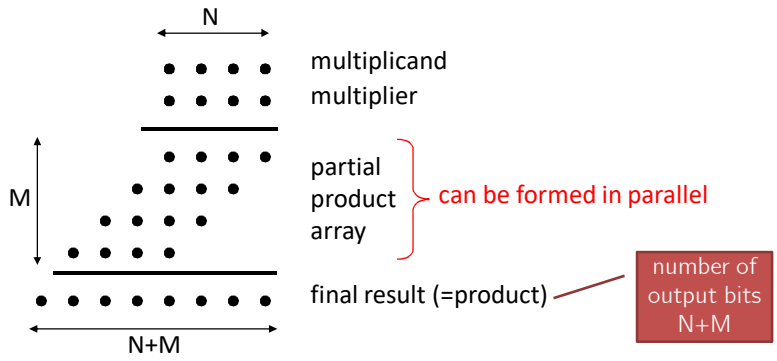
902

Another key building block

Multiplier circuits

903

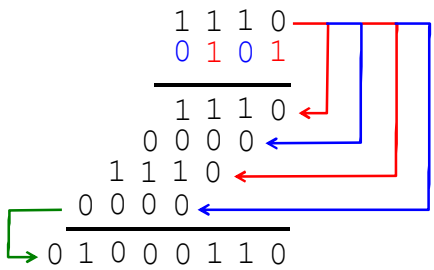
Multiplication as repeated additions



- Final result (product) is obtained through multi-operand addition

904

Example



if multiplier 1 then use multiplicand

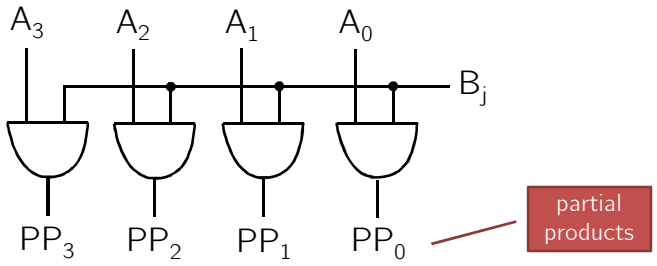
if multiplier 0 then not use multiplicand

add all partial products

- Produce M partial products of N-bit
- Sum these to produce M+N bit product

905

Partial product generation



- In most cases, the partial product array has many zero rows that have no impact on result

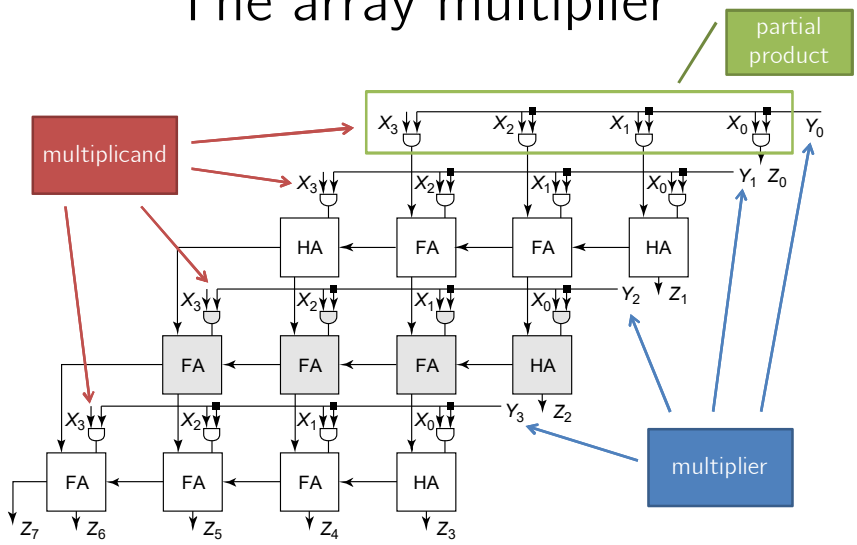
906

Booth's recoding method

- Goal: Reduce number of generated PPs
- Example:
  - Assume multiplier is 01111110
  - Generates 6 non-zero PPs
  - Recode multiplier to 10000$\underline{1}$0 — $\underline{1}$ indicates -1
  - Recoded number has only 2 non-zero PPs
- Booth's recoding method reduces the number of non-zero PPs by half
  - Lower area and faster → but complicated ☹

907

The array multiplier

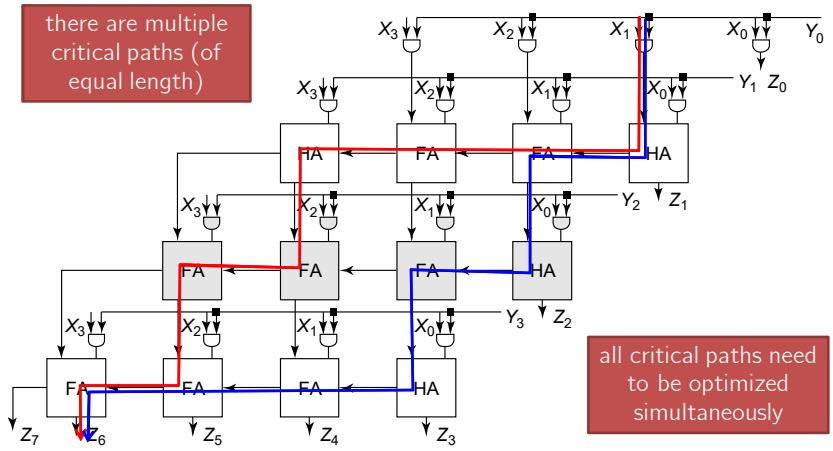


908

Image taken from: Digital Integrated Circuits (2nd Edition) by Rabaey, Chandrakasan, Nikolic

Critical paths of array multiplier

there are multiple critical paths (of equal length)



all critical paths need to be optimized simultaneously

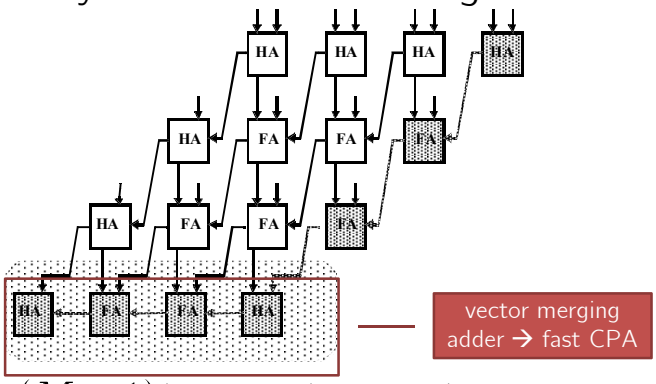
$$t_{mult} \approx ((N-1)+(M-2))t_{carry} + (M-1)t_{sum} + t_{and}$$

909

Image taken from: Digital Integrated Circuits (2nd Edition) by Rabaey, Chandrakasan, Nikolic

Carry-save multiplier

- Pass the multiplication results diagonally in the array instead of on the right



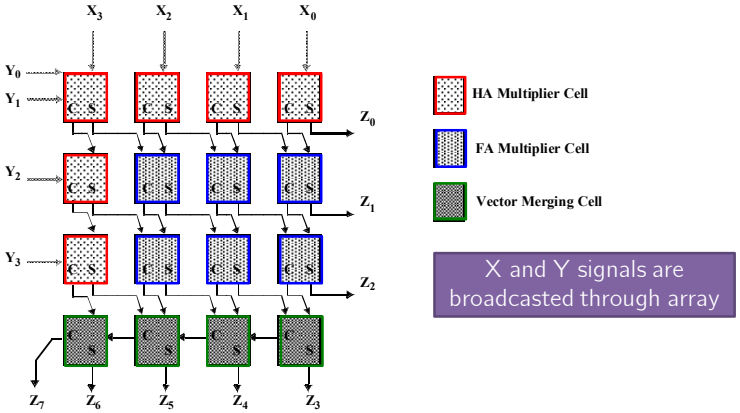
$$t_{mult} \approx (M-1)t_{carry} + t_{CPA} + t_{and}$$

910

Image taken from: Digital Integrated Circuits (2nd Edition) by Rabaey, Chandrakasan, Nikolic

Carry-save multiplier (cont'd)

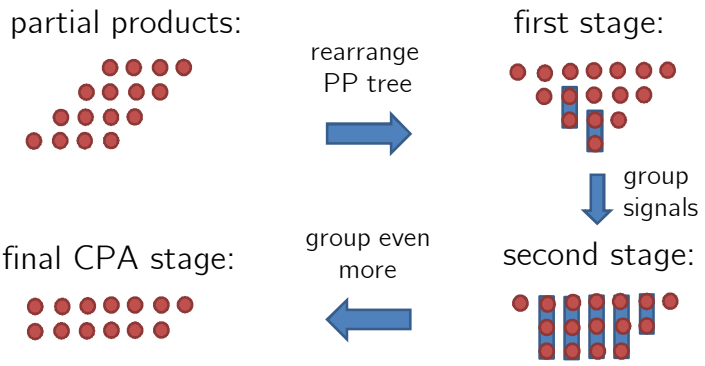
- Improved floor-plan → optimized for regularity



911

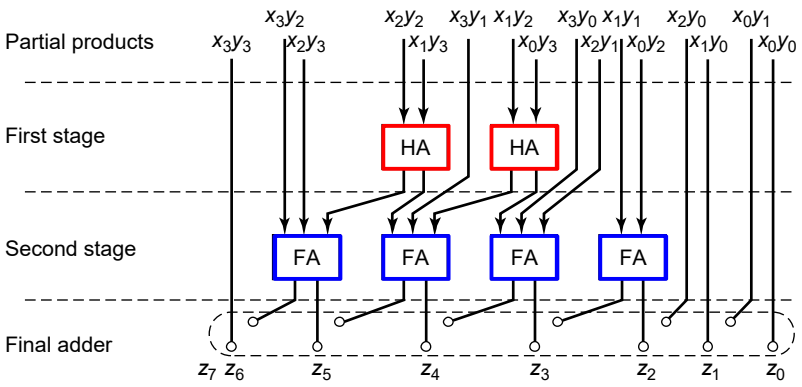
Remember the Wallace tree adder

- Faster PP summation with Wallace tree



912

Wallace tree multiplier



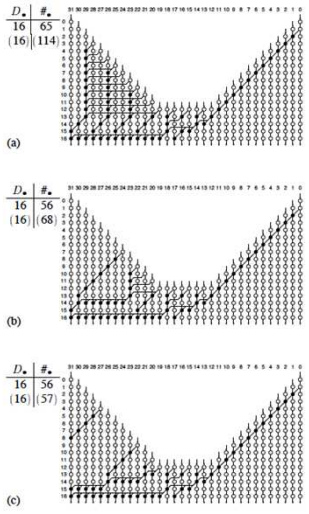
- Propagation delay through tree T=O(log(N))
- Final adder needs to be chosen carefully, WHY?

913

Image taken from: *Design and Analysis of Low Power Compressors* by Karthick, Karthika, Valarmathy

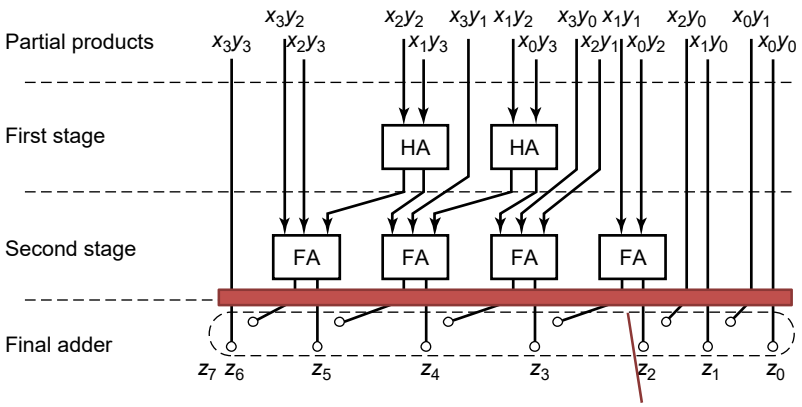
Remember these tree adders?

- Critical signals are those for the middle output bits
- LSB and MSB arrive early
- One can design optimized prefix-tree adders for multiplier outputs



914

Wallace tree multiplier (cont'd)



- Design can be pipelined
  - Common approach in DSPs

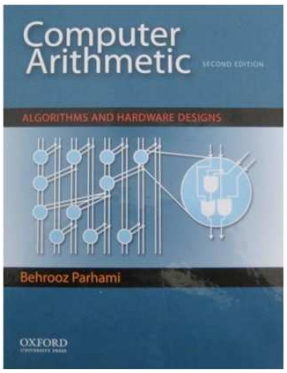
915

Multiplier summary

- Optimization goals different to adders
  - Multiple critical paths
- Different techniques to make it fast
  - Use Booth's recoding
  - Use carry save adder (CSA)
  - Use Wallace tree (or Dadda tree)
  - Wisely choose final adder
  - Pipelining

The Swiss Army knife

CORDIC*



*B. Parhami, "Computer Arithmetic," 2nd Ed., Oxford Univ. Press, 2010

917

How to compute other operations?

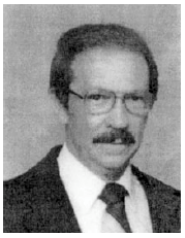
- Trigonometric/hyperbolic functions
  - sin(z), cos(z), and $\tan^{-1}(z)$
  - sinh(z), cosh(z), and $\tanh^{-1}(z)$
- Division and multiplication: x/y and x*z
- Norms: sqrt(x^2+y^2) or sqrt(x^2-y^2)
- Angle of 2D vector, Givens rotation, etc....

CORDIC = coordinate rotation digital computers

918

The inventor

- Jack E. Volder
  - flight engineer during WW2
  - 1956 → replace analog computer of B58 bomber with digital computer
  - 250 kHz clock rate (!)
  - Found this in a formula book:

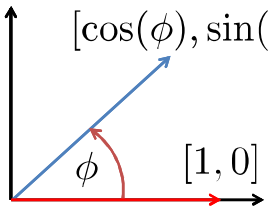


$$K_n R \sin(\theta \pm \phi) = R \sin(\theta) \pm 2^{-n} R \cos(\theta)$$
$$K_n R \cos(\theta \pm \phi) = R \cos(\theta) \mp 2^{-n} R \sin(\theta)$$

J. E. Volder, "The Birth of CORDIC," J. VLSI Sig. Proc., 2000

919

Compute 2D rotations



- Goal: rotate unit vector [1,0] by ϕ

- Idea: instead of performing the rotation at once, perform a series of "pseudo rotations"
- Pseudo-rotations are hardware friendly

920

Givens rotation

- 2D rotation:

start vector

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix}}_{\text{Givens rotation: } \mathbf{R}(\phi)} \begin{bmatrix} x \\ y \end{bmatrix}$$

result of rotation

- Rewrite the Givens rotation matrix:

$$\mathbf{R}(\phi) = \cos(\phi) \begin{bmatrix} 1 & -\tan(\phi) \\ \tan(\phi) & 1 \end{bmatrix}$$

921

Givens rotation (cont')

- Rewrite the rotation matrix even more:

$$\mathbf{R}(\phi) = \frac{1}{\sqrt{1+\tan^2(\phi)}} \begin{bmatrix} 1 & -\tan(\phi) \\ \tan(\phi) & 1 \end{bmatrix}$$

- Decompose the angle ϕ into K so-called micro rotations:

$$\phi = \phi_0 + \phi_1 + \cdots + \phi_{K-1}$$

922

Micro rotation

- Restrict micro-rotation angles to satisfy:

$$\tan(\phi_i) = \pm 2^{-i}$$

possible rotation angles are
i=0 → ±45.0°
i=1 → ±26.6°
i=2 → ±14.0°
…

- Micro-rotation matrix is given by

$$\mathbf{R}(\sigma_i, i) = \underbrace{\frac{1}{\sqrt{1+2^{-2i}}}}_{C_i} \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \qquad \sigma_i \in \{-1, +1\}$$

923

Pseudo rotation

- Idea: Approximate Given rotation with a series of micro rotations: $\mathbf{R}(\phi) \approx \prod_{i=0}^{K-1} \mathbf{R}(\sigma_i, i)$
- Ignore scaling C_i during pseudo rotations

$$\prod_{i=0}^{K} \mathbf{R}(\sigma_i, i) = \prod_{i=0}^{K-1} C_i \prod_{i=0}^{K-1} \mathbf{P}(\sigma_i, i)$$

- Pseudo rotation:

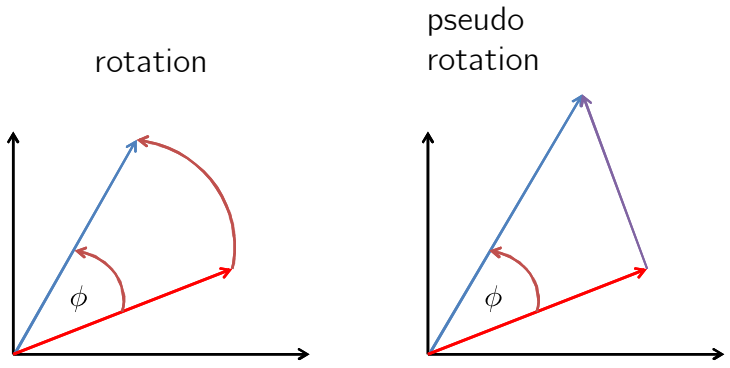
$$\mathbf{P}(\sigma_i, i) = \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix}$$

$\sigma_i \in \{-1, +1\}$

requires only shifts and additions

924

What is a pseudo rotation?

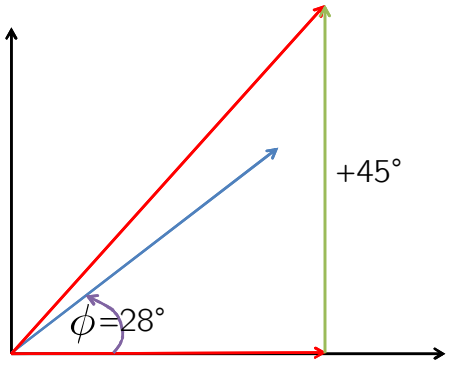


- Pseudo-ration does not preserve length of vector

925

CORDIC example

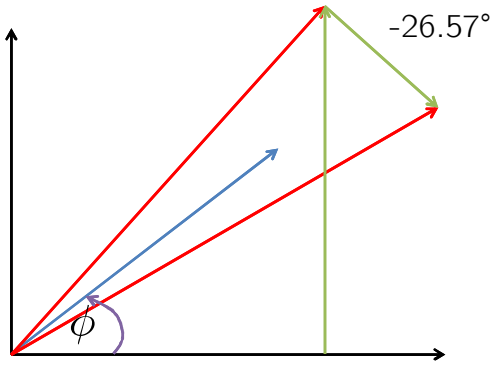
- Rotate a vector from [1,0] to desired angle using pseudo rotations:



926

CORDIC example (cont'd)

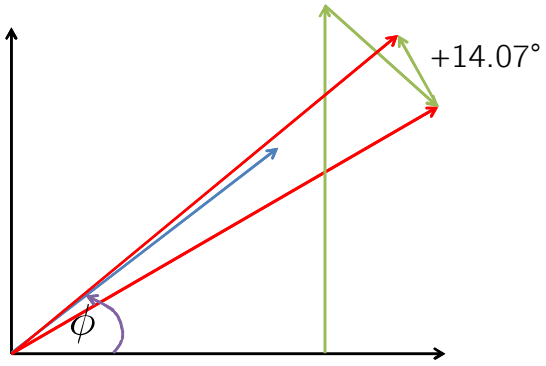
- Rotate a vector from [1,0] to desired angle using pseudo rotations:



927

CORDIC example (cont'd)

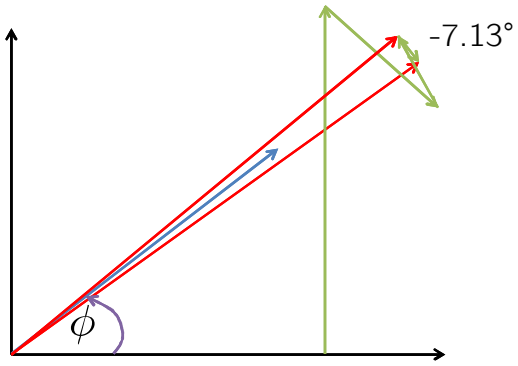
- Rotate a vector from [1,0] to desired angle using pseudo rotations:



928

CORDIC example (cont'd)

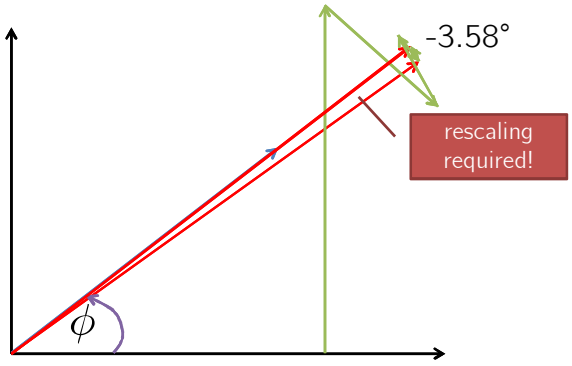
- Rotate a vector from [1,0] to desired angle using pseudo rotations:



929

CORDIC example (cont'd)

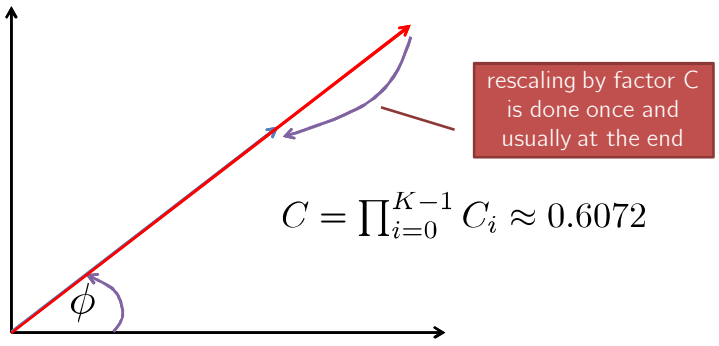
- Rotate a vector from [1,0] to desired angle using pseudo rotations:



930

CORDIC example (cont'd)

- Length of vector increases by about 1.647 due to pseudo-rotations



931

How to determine angles ϕ_i ?

- Only need to chose: $\sigma_i \in \{-1, +1\}$
- Keep track of how much vector was pseudo-rotated so far:

rotation angles pre-computed and stored in a lookup table

$$z_i = z_{i-1} - \sigma_i \phi_i \qquad \phi_i = \arctan(2^{-i})$$

- Select σ_i such that z_i is closer to ϕ than z_{i-1}
  - Can be done with simple sign comparisons

932

CORDIC algorithm summary

- Rewrite 2D rotation as

$$\mathbf{x}' = \mathbf{R}(\phi)\mathbf{x} \quad \Rightarrow \quad \mathbf{x}' = \prod_{i=0}^{K-1} \mathbf{R}(\phi_i)\mathbf{x}$$

- Perform series of pseudo-rotations instead of Givens rotations:

$$\tilde{\mathbf{x}} = \prod_{i=0}^{K-1} \mathbf{P}(\sigma_i, i)\mathbf{x} \qquad \mathbf{P}(\sigma_i, i) = \begin{bmatrix} 1 & -\sigma_i 2^{-1} \\ \sigma_i 2^{-1} & 1 \end{bmatrix}$$

- Rescale result: $\mathbf{x}' \approx C\tilde{\mathbf{x}}$

933

CORDIC properties

- K bit precision requires ~K CORDIC pseudo-rotations
- Achievable rotation angles between -99.7° to 99.7°
  - full range can be obtained by adding a 180° rotation (=mirror at origin if necessary)
- Can be implemented efficiently in VLSI

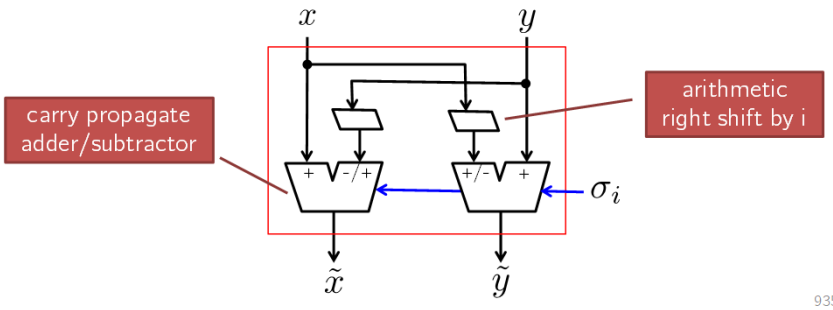
i	deg.	rad.
0	45.00	0.785
1	26.57	0.464
2	14.04	0.245
3	7.13	0.124
4	3.58	0.062
5	1.79	0.031
6	0.90	0.016
7	0.45	0.008
8	0.22	0.004
9	0.11	0.002
10	...	...

934

Pseudo rotations = hardware friendly

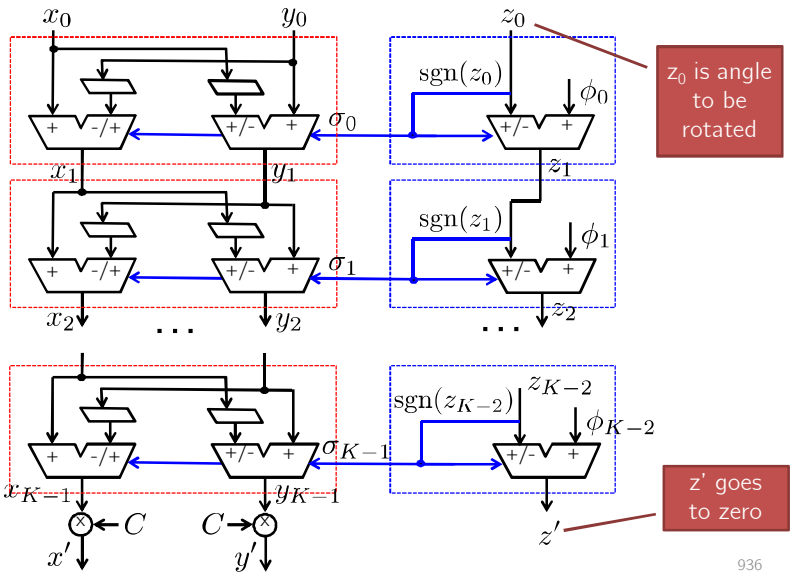
$$\begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} = \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- VLSI implementation of pseudo rotation:



935

VLSI architecture: rotation CORDIC



936

Universal CORDIC*

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1 & -\mu d_i 2^{-i} \\ d_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \qquad z_{i+1} = z_i - d_i \alpha_i$$

Mode	Rotation	Vectoring
	$d_i = \mathrm{sgn}\ (z_i), \quad z \to 0$	$d_i = -\mathrm{sgn}\ (x_i y_i), \quad y \to 0$
Circular $\mu = 1$ $\alpha_i = \tan^{-1} 2^{-i}$	$x \to K(x\cos z - y\sin z)$, $y \to K(y\cos z + x\sin z)$, $z \to 0$	$x \to K\sqrt{x^2+y^2}$, $y \to 0$, $z \to z + \tan^{-1}(y/x)$
Linear $\mu = 0$ $\alpha_i = 2^{-i}$	$x \to x$, $y \to y + xz$, $z \to 0$	$x \to x$, $y \to 0$, $z \to z + y/x$
Hyperbolic $\mu = -1$ $\alpha_i = \tanh^{-1} 2^{-i}$	$x \to K'(x\cosh z - y\sinh z)$, $y \to K'(y\cosh z + x\sinh z)$, $z \to 0$	$x \to K\sqrt{x^2-y^2}$, $y \to 0$, $z \to z + \tanh^{-1}(y/x)$

*B. Parhami, "Computer Arithmetic," 2nd Ed., Oxford Univ. Press, 2010

937

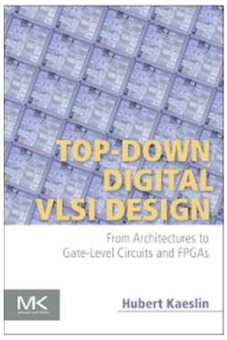
CORDIC summary

- Essentially consists of shifts and adds
- CORDIC has multiple modes
  - All share same architecture
- Used in VLSI circuits for communication systems, array processing, etc.
- Can be made faster and smaller using
  - Carry save adders
  - Architecture transforms

938

Play with the area/delay trade-off

Architecture transforms*



*H. Kaeslin, "Top-Down Digital VLSI Design," Morgan Kaufmann, 2014

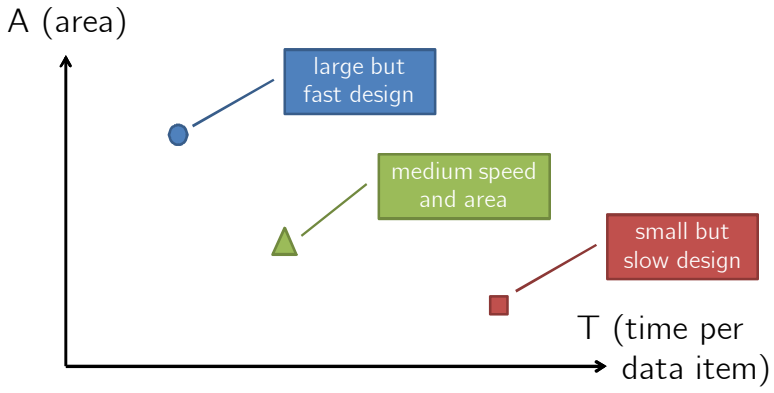
939

Area/delay trade-off

- In most cases, one can make a VLSI design larger but faster or smaller but slower ☹
  - Trade-off between area and propagation delay
- Ultimate goal: **smaller and faster**
- Architecture transforms very useful to find suitable architecture in the trade-off space
- Some transforms improve **area and delay!**

940

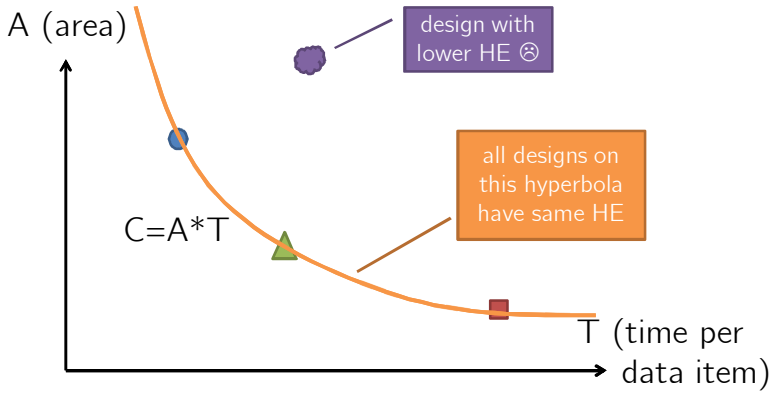
The AT diagram



- Hardware efficiency: HE = A*T

941

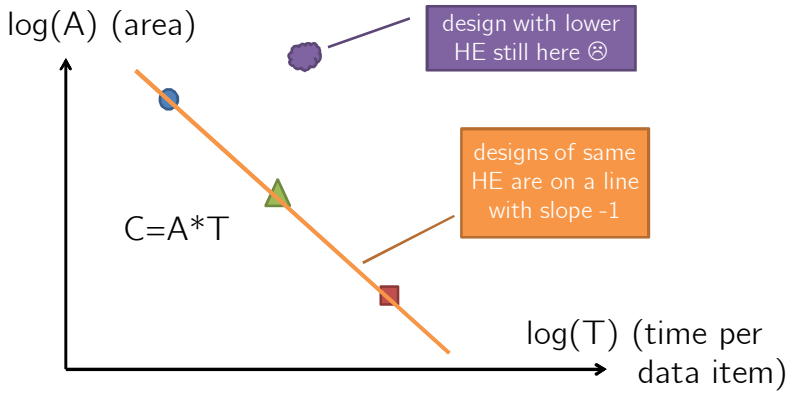
Hardware efficiency (HE)



- Constant hardware efficiency: C=A*T 

942

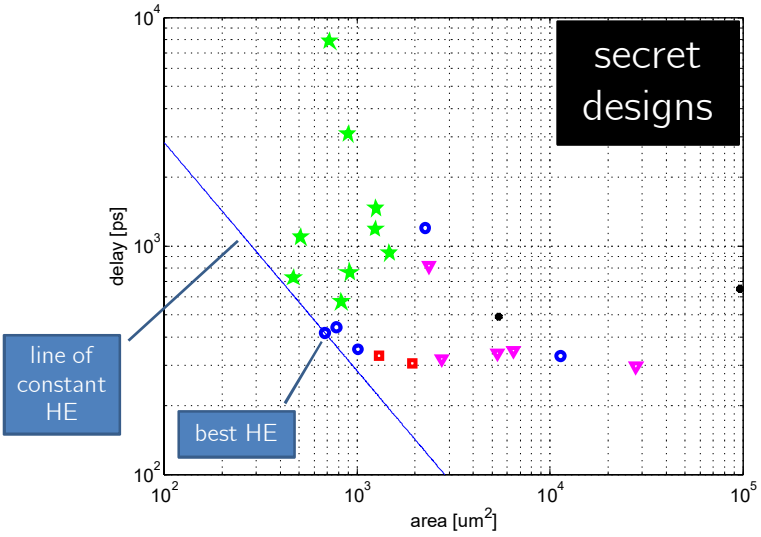
Logarithmic AT diagram



- Reason: log(C)=log(A)+log(T)
- Helpful to visualize effect of transforms

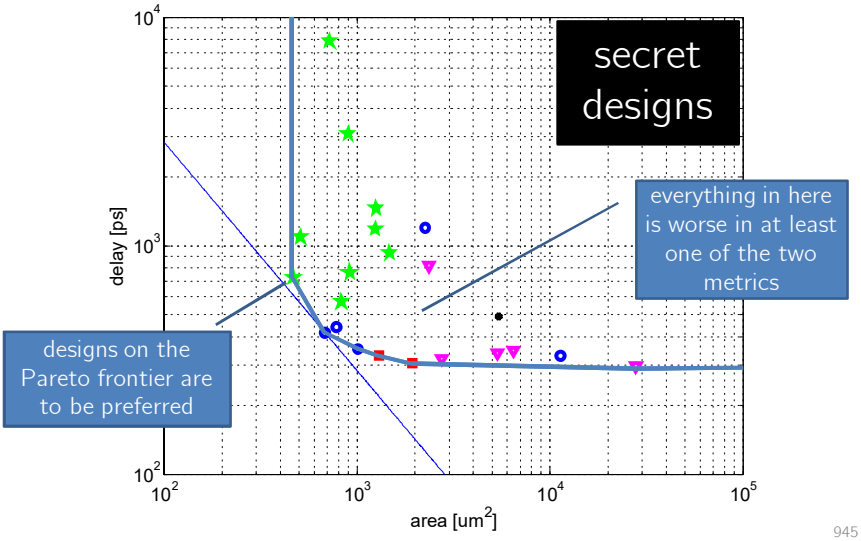
943

A real-world example



944

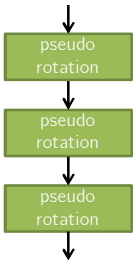
Pareto optimality



945

Architecture transforms

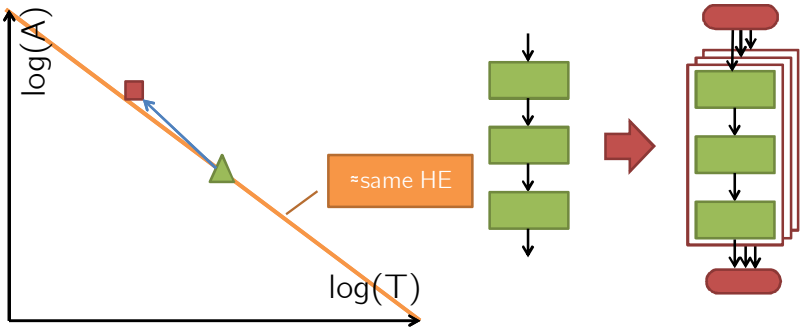
- We use a CORDIC-like architecture to show the most important transforms
- Same ideas apply to almost all VLSI designs



- Isomorphic architecture → every operation has its own dedicated circuit
- We will improve the HE of CORDICs!

946

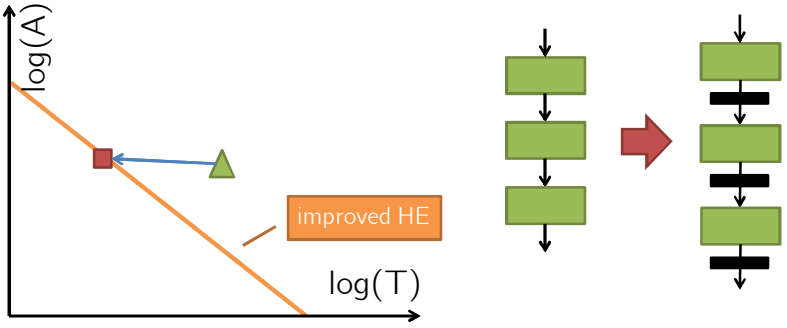
Transform 1: Replication



- Idea: Use N instances of the same unit
- Requires distribution and collection units
- $A'=A*N+A_{dist}+A_{coll}$, $T'=T/N+T_{dist}+T_{coll}$

947

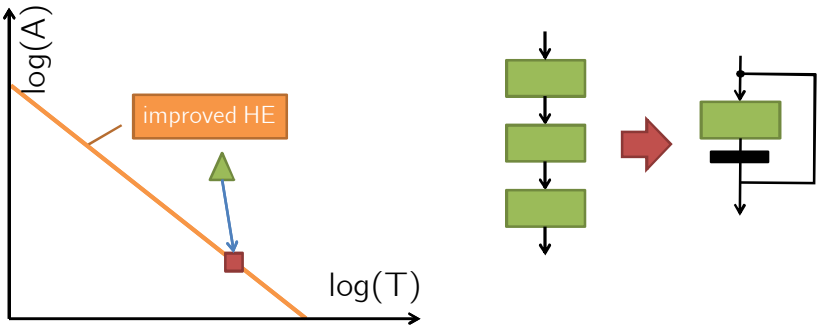
T2: Pipelining



- Insert N flip-flops into datapath
- $A' = A + N * A_{ff}$, $T' \approx T/N + T_{ff}$
- Pipelining improves hardware efficiency!

948

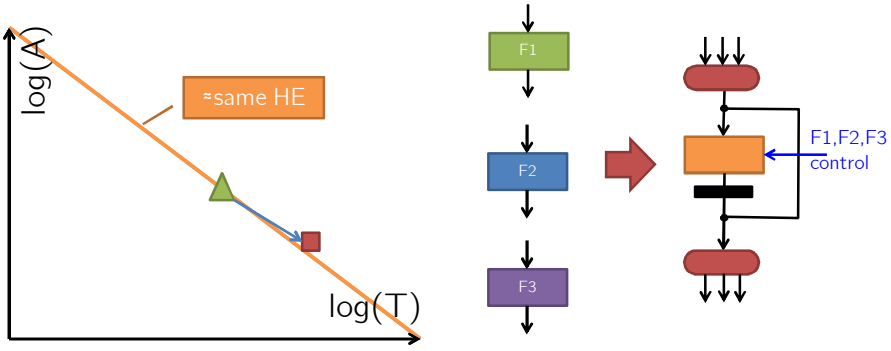
T3: Iterative decomposition



- Share resources & step-by-step execution
- $A' \approx A/N + A_{ff}$, $T' \approx T + N * T_{ff}$
- Iterative decomposition improves HE!

949

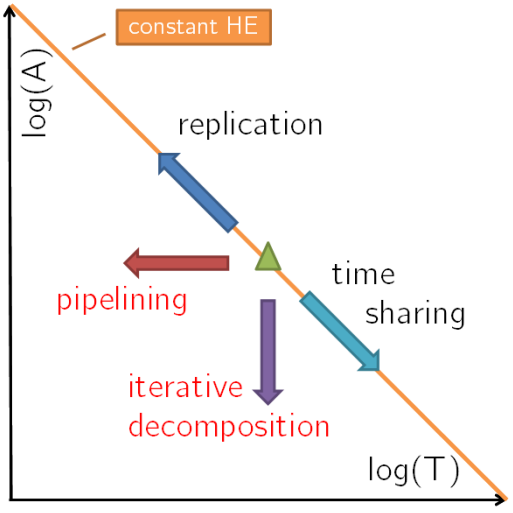
T4: Time sharing (or resource sharing/multiplexing)



- Process N different tasks on one unit
- Requires distributor, collector, and control unit
- $A' \approx A/N + A_{ff} + A_{dist} + A_{coll}$, $T' \approx T*N + T_{ff}$

950

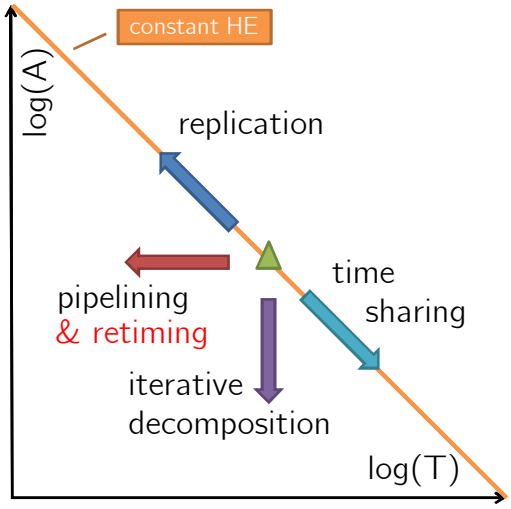
Summary of architecture transforms



- Important:
  - Don't forget overhead of each transform
  - Possible transforms depend on application

951

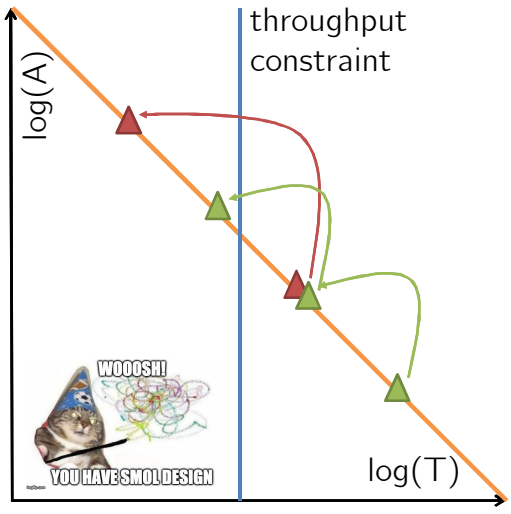
Where is retiming?



- Retiming reduces delay
- Retiming can reduce or increase area
- Effects of retiming usually smaller than pipelining

952

Smaller designs can be preferable



- Consider an application that allows replication
- Small designs offer higher granularity
- Replicate to achieve desired throughput

953