Lecture 10: Bit-Level Arithmetic Architectures

Sources: Peter Nilsson

Outline

- Why?
- Number Representation
- Multiplication
  - Parallel Multipliers
  - Parallel carry-ripple array multiplier
  - Parallel carry-save array multiplier
  - Booth Multiplier
  - Bit serial multipliers
  - Bit-Serial FIR Filter (see textbook, self-study)
- Canonic Signed Digit Arithmetic
- Distributed Arithmetic
- Self-study
  - Newton Raphson: efficient for computing 1/d
  - CORDIC Algorithm

Why the study is important?

- “Computations” in SW and HW are based on the bit-level arithmetic
- Efficient implementation of the arithmetic functions (units) is one of the key factors in DSP

Number Representation

- Unsigned number representation: Fixed radix (base) systems

The digits $a \in \{0, 1, 2, \ldots\ r-1\}$ **in a radix** r **system:**

$$\sum_{i=k-1}^{-l} \underbrace{r^i}_{\text{weight}} \times \underbrace{a_i}_{\text{digit}} =$$

$$= r^{k-1} a_{k-1} + r^{k-2} a_{k-2} \cdots r^1 a_1 + r^0 a_0 + r^{-1} a_{-1} \cdots r^{-l} a_{-l}$$

described in a fixed point positional number system:

$$a_i\, a_{i-1} \cdots a_1\, a_0 \cdot \underbrace{a_{-1} \cdots a_{-l}}_{\text{Fractional part}}$$

Number Representation

- Example of the unsigned number representation

$$\sum_{i=k-1}^{-l} 10^i a_i = \quad \{a \in \{0, 1, 2, \ldots 9\} \text{ in radix } 10\}$$

$$= 10^{k-1} a_{k-1} + 10^{k-2} a_{k-2} \cdots 10 a_1 + a_0 + 10^{-1} a_{-1} \cdots 10^{-l} a_{-l}$$

$$\sum_{i=k-1}^{-l} 2^i a_i = \quad \{a \in \{0, 1\} \text{ in radix } 2\}$$

$$= 2^{k-1} a_{k-1} + 2^{k-2} a_{i-2} \cdots 2 a_1 + a_0 + 2^{-1} a_{-1} \cdots 2^{-l} a_{-l}$$

Signed Digit Number Representation

- Example of signed number representation

The digits $a \in \{-\alpha, \ldots 0, \ \ldots \ r-1-\alpha\}$ **in a radix** r **system:**

$$\sum_{i=k}^{-l} r^i \times a_k$$

Example Radix 10: $a \in \{-4, -3, \ldots 0, \ \ldots \ 4, 5\}$

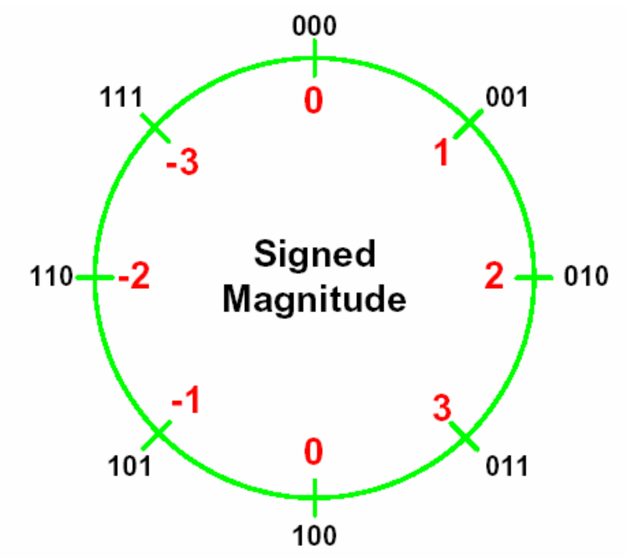
$$(3 \quad -1 \quad 5)_{10} = 10^2 \times 3 - 10^1 \times 1 + 10^0 5 = 300 - 10 + 5 = 295$$

$$(3 \quad .-1 \quad 5)_{10} = 3 - 10^{-1} \times 1 + 10^{-2} \times 5 = 3 - 0.1 + 0.05 = 2.95$$

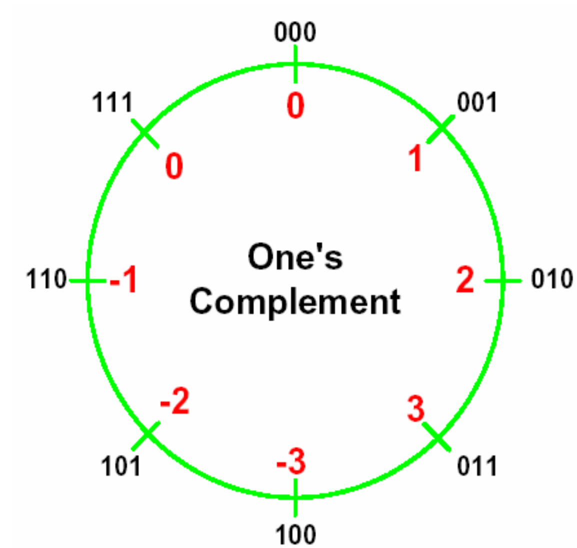
Modified Booth's recoding - a
signed digit radix 4 representation

Number Representation

- Signed Number Representation
  - Sign magnitude
  - One's complement
  - Two's complement
- Signed magnitude: MSB(sign bit)+ other bits (magnitude)

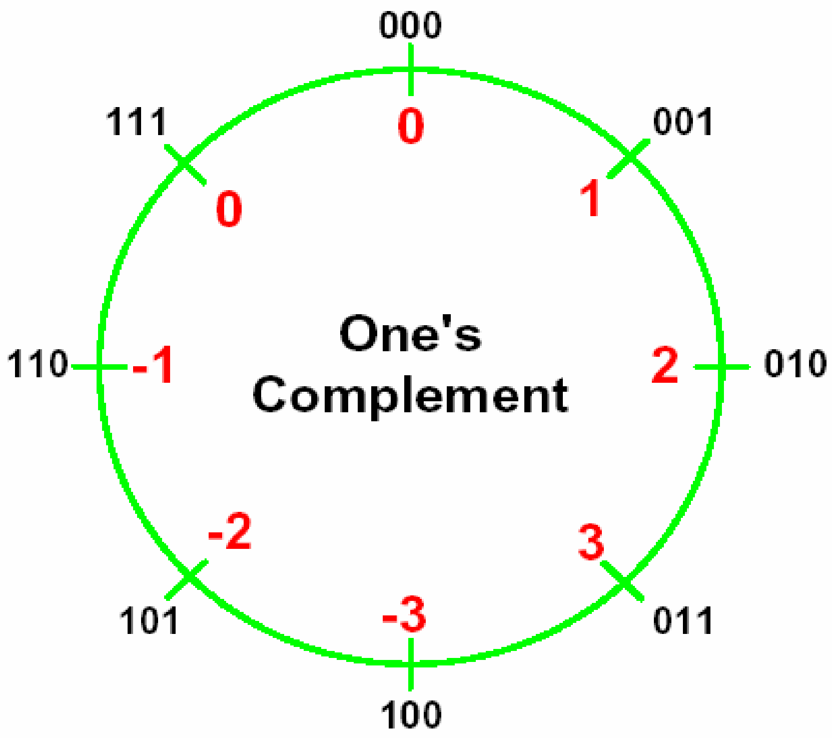


Bad: two zeros
Good: Low Power than 1's complement (next slide) why?



Number Representation

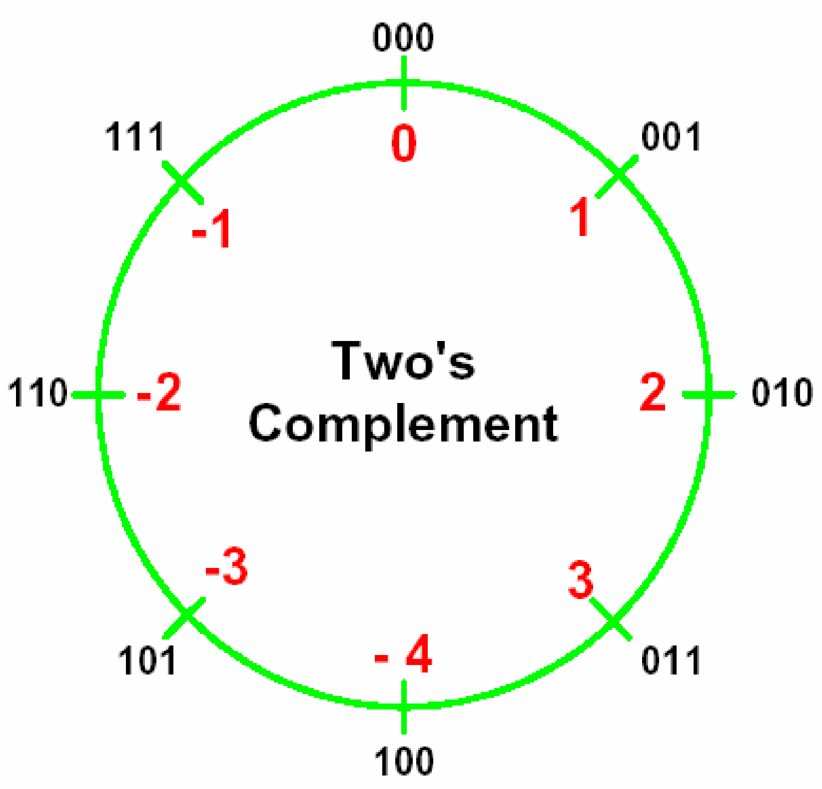
- One's complement: signed numbers by inverting (complement)
  - MSB (signed bit) + others (magnitude if sign bit =1, one's complement if sign bit=0)



Bad: two zeros
Good: easy to convert to negative

Number Representation

- Two's Complement : MSB (sign bit) + others (magnitude if the sign bit = 0, (one's complement +1) if the sign bit =1)



Bad: not so easy to convert to negative
Good: easy addition, one zeros

Number Representation

- Two's Complement

$$110.01 = -2^{3-1}\cdot 1 + 2^{3-2}\cdot 1 + 2^{3-3}\cdot 0 + 2^{-1}\cdot 0 + 2^{-2}\cdot 1$$
$$= -4 + 2 + 0.25 = -1.75$$
$$= -(01.10$$
$$+\,00.01)$$
$$= -(01.11) = -(1+0.5+0.25) = -1.75$$

The digits $a \in \{0,1\}$ in a radix 2 system:

$$-2^{k-1}\times a_{k-1} + \sum_{i=k-2}^{-l} 2^i \times a_i =$$

$$= -2^{k-1}a_{k-1} + 2^{k-2}a_{k-2}\cdots 2^1 a_1 + 2^0 a_0 + 2^{-1}a_{-1}\cdots 2^{-l}a_{-l}$$

described in a fixed point positional number system:

$$a_{k-1}\,a_{k-2}\cdots a_1\,a_0\,.\,\underbrace{a_{-1}\cdots a_{-l}}_{\text{Fractional part}}$$

Sign Bit

Number Representation

- Sign Extension in Two's Complement

$$\boxed{-2^{k-1}a_{k-1}} + 2^{k-2}a_{k-2}\cdots 2a_1 + a_0 =$$

$$\boxed{-2^{k}a_{k-1} + 2^{k-1}a_{k-1}} + 2^{k-2}a_{k-2}\cdots 2a_1 + a_0 =$$

$$-2^{k+1}a_{k-1} + 2^{k}a_{k-1} + 2^{k-1}a_{k-1} + 2^{k-2}a_{k-2}\cdots 2a_1 + a_0$$

$$\begin{aligned}
-2^{k-1}\cdot a_{k-1} &= -2^{k}\cdot a_{k-1} + 2^{k-1}\cdot a_{k-1} \\
&= (-2^{k} + 2^{k-1})\cdot a_{k-1} \\
&= ((-2\cdot 2^{k})/2 + 2^{k}/2)\cdot a_{k-1} \\
&= (-2^{k})/2 = -2^{k-1}\cdot a_{k-1}
\end{aligned}$$

Example:

$$10010 = 11\underline{0010} = 111\underline{0010} = 1111\underline{0010} = \cdots$$

$$00010 = 000010 = 0000010 = 00000010 = \cdots$$

Unsigned Multiplication

$$P = A \times B = A \times \sum_{i=3}^{0} 2^{i} \times b_i =$$

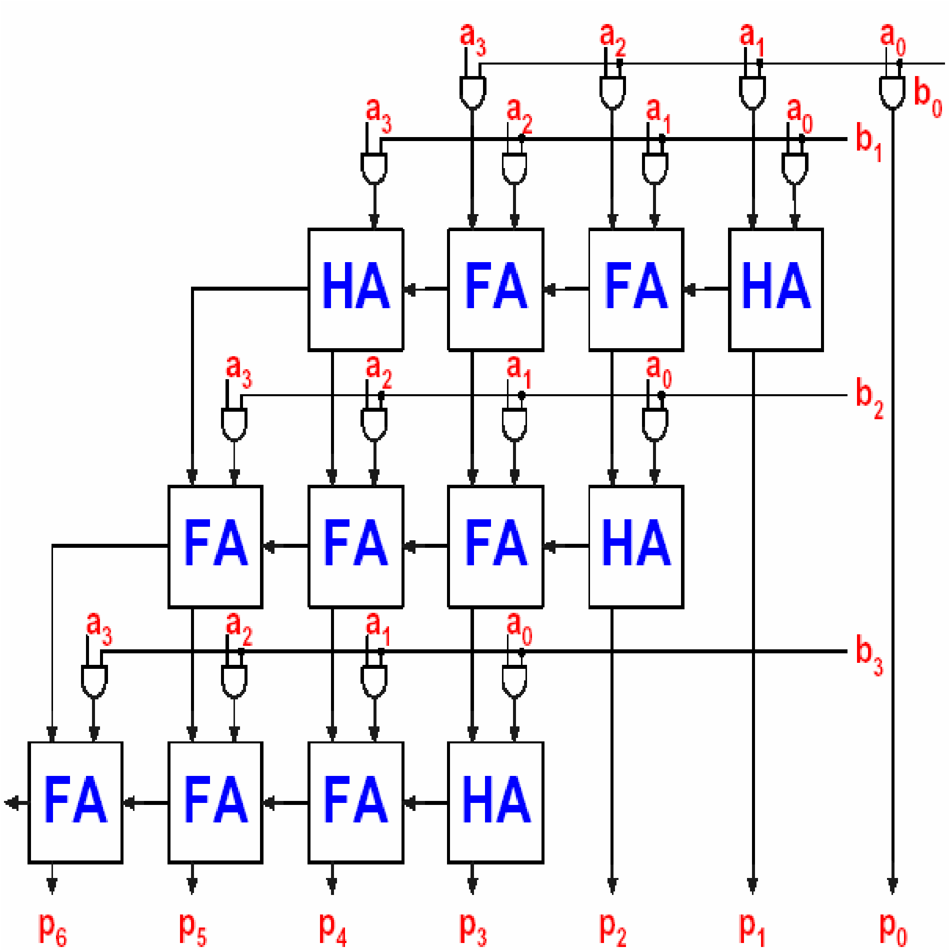
$$= A \times 2^3 b_3 + A \times 2^2 b_2 + A \times 2^1 b_1 + A \times 2^0 b_0$$

			a_3	a_2	a_1	a_0
			b_3	B_2	b_1	b_0
			$a_3 b_0$	$a_2 b_0$	$a_1 b_0$	$a_0 b_0$
		$a_3 b_1$	$a_2 b_1$	$a_1 b_1$	$a_0 b_1$	
	$a_3 b_2$	$a_2 b_2$	$a_1 b_2$	$a_0 b_2$		
$a_3 b_3$	$a_2 b_3$	$a_1 b_3$	$a_0 b_3$			
p_6	p_5	p_4	p_3	p_2	p_1	p_0

Shifted partial products

Array Multiplier

- Unsigned arithmetic



$$P = A \times B = A \times \sum_{i=3}^{0} 2^i \times b_i =$$

$$= A \times 2^3 b_3 + A \times 2^2 b_2 + A \times 2^1 b_1 + A \times 2^0 b_0$$

			a_3	a_2	a_1	a_0
			b_3	B_2	b_1	b_0
			$a_3 b_0$	$a_2 b_0$	$a_1 b_0$	$a_0 b_0$
		$a_3 b_1$	$a_2 b_1$	$a_1 b_1$	$a_0 b_1$	
	$a_3 b_2$	$a_2 b_2$	$a_1 b_2$	$a_0 b_2$		
$a_3 b_3$	$a_2 b_3$	$a_1 b_3$	$a_0 b_3$			
p_6	p_5	p_4	p_3	p_2	p_1	p_0

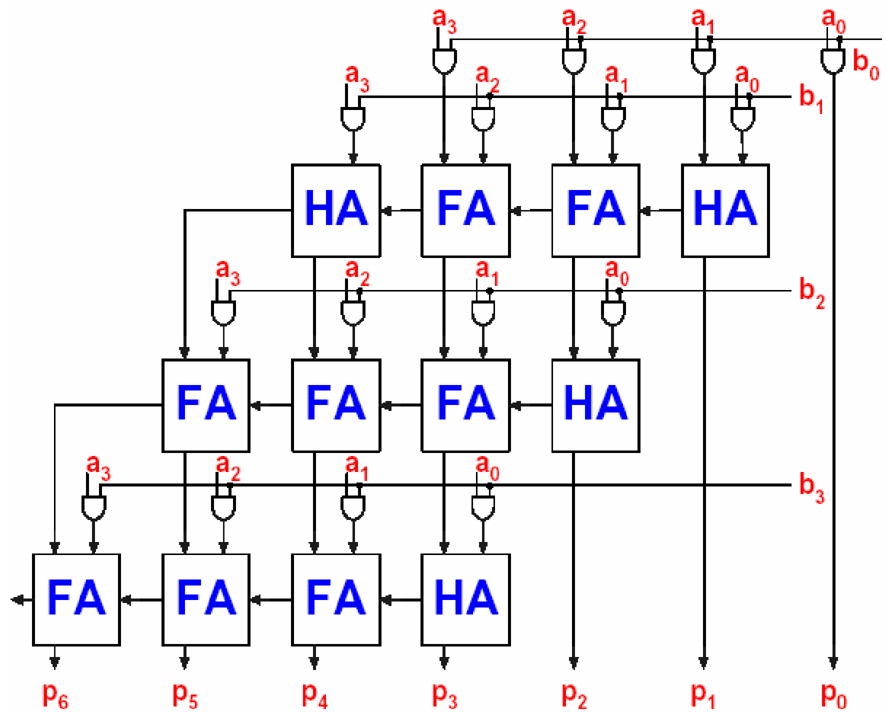
Shifted partial products

Unsigned Multiplication

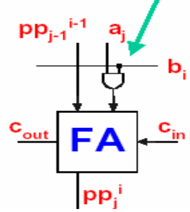
$$= A\times 2^3 b_3 + A\times 2^2 b_2 + A\times 2^1 b_1 + A\times 2^0 b_0$$

			a_3	a_2	a_1	a_0
			b_3	b_2	b_1	b_0
			a_3b_0	a_2b_0	a_1b_0	a_0b_0
		a_3b_1	a_2b_1	a_1b_1	a_0b_1	
		pp_3^1	pp_2^1	pp_1^1	pp_0^1	
	a_3b_2	a_2b_2	a_1b_2	a_0b_2		
	pp_3^2	pp_2^2	pp_1^2	pp_0^2		
a_3b_3	a_2b_3	a_1b_3	a_0b_3			
p_6	p_5	p_4	p_3	p_2	p_1	p_0

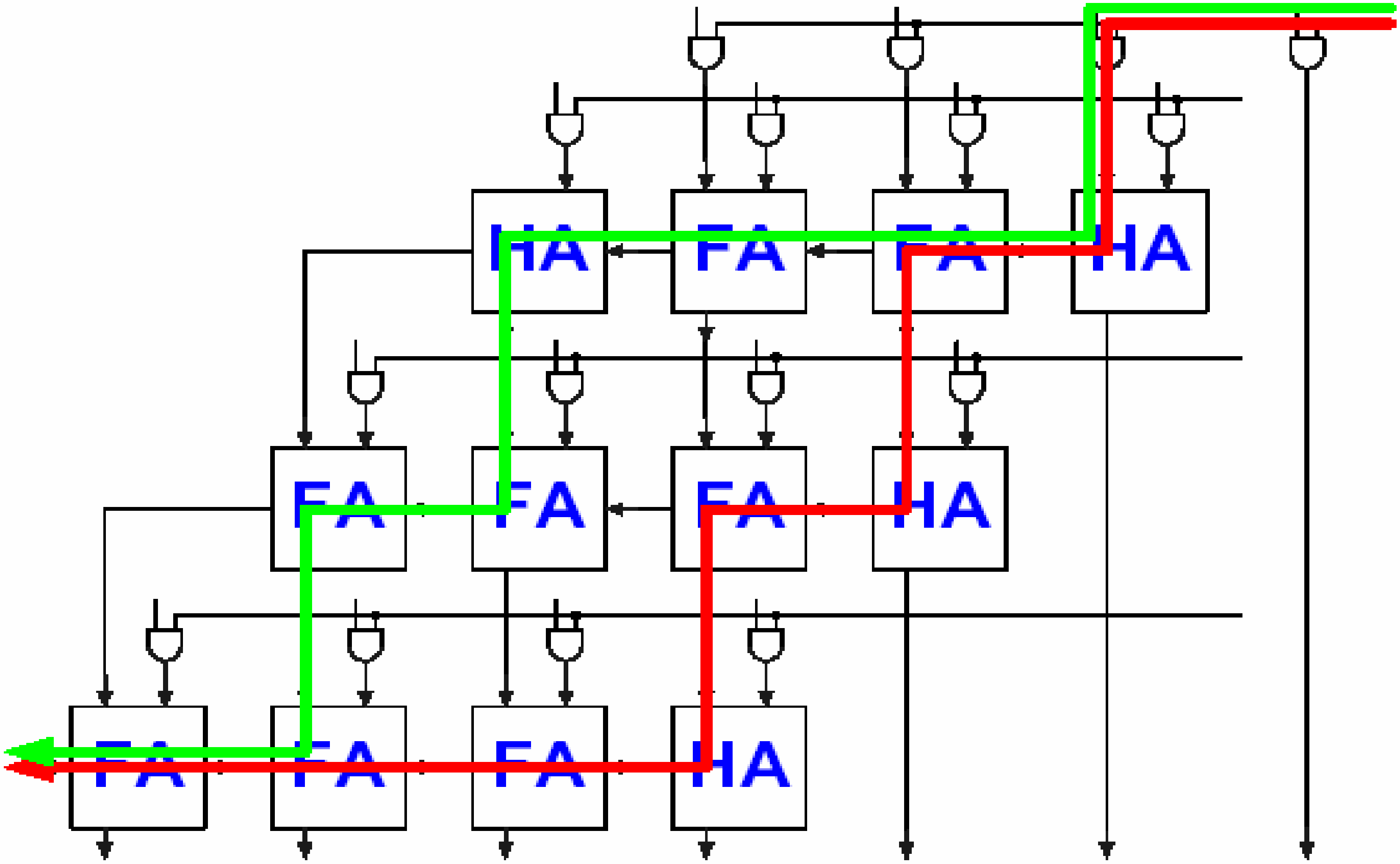
Rows in Multiplier



Bit Multiplication



Array Multiplier: Critical Path



Two's Complement (Horners Rule)

$$A \times B =$$

$$b_0 2^0 \times (-2^3 a_3 + 2^2 a_2 + 2^1 a_1 + a_0) +$$

$$b_1 2^1 \times (-2^3 a_3 + 2^2 a_2 + 2^1 a_1 + a_0) +$$

$$b_2 2^2 \times (-2^3 a_3 + 2^2 a_2 + 2^1 a_1 + a_0) +$$

$$-b_3 2^3 \times (-2^3 a_3 + 2^2 a_2 + 2^1 a_1 + a_0)$$

Solved by sign extension

Need to be rewritten

Multiplication (Horners Rule)

$$-b_3 2^3 \times (-2^3 a_3 + 2^2 a_2 + 2^1 a_1 + a_0) =$$

LSB

$$= b_3 2^3 \times (2^3 a_3 - 2^2 a_2 - 2^1 a_1 - a_0) =$$

$$= b_3 2^3 \times (-2^3 \overline{a_3} + 2^2 \overline{a_2} + 2^1 \overline{a_1} + \overline{a_0} + 1) =$$

$$= b_3 2^3 \times (-2^3 \overline{a_3} + 2^2 \overline{a_2} + 2^1 \overline{a_1} + \overline{a_0}) + b_3 2^3$$

Complemented

Multiplication (Horners Rule)

$$
\begin{array}{ccccccc}
 & & & a_3 & a_2 & a_1 & a_0 \\
 & & & b_3 & b_2 & b_1 & b_0 \\
\hline
 & & & -a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 \\
 & & -a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & \\
 & -a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & \\
-\bar{a}_3b_3 & \bar{a}_2b_3 & \bar{a}_1b_3 & \bar{a}_0b_3 & & & \\
 & & & b_3 & & & \\
\hline
p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}
$$

$[A \times b_0]$

$[A \times b_1] \times 2$

$$b_3 2^3 \times (-2^3 \overline{a_3} + 2^2 \overline{a_2} + 2^1 \overline{a_1} + \overline{a_0}) + b_3 2^3$$

Multiplication (Horners Rule)

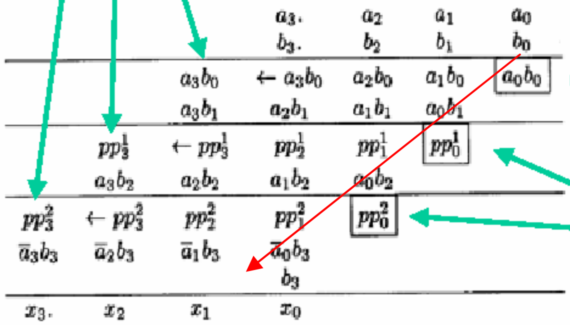
Negative MSBs solved with sign extension, one in each partial product

			$a_3.$	a_2	a_1	a_0
			$b_3.$	b_2	b_1	b_0
		a_3b_0	$\leftarrow a_3b_0$	a_2b_0	a_1b_0	$\boxed{a_0b_0}$
		a_3b_1	a_2b_1	a_1b_1	a_0b_1	
	pp_3^1	$\leftarrow pp_3^1$	pp_2^1	pp_1^1	$\boxed{pp_0^1}$	
	a_3b_2	a_2b_2	a_1b_2	a_0b_2		
pp_3^2	$\leftarrow pp_3^2$	pp_2^2	pp_1^2	$\boxed{pp_0^2}$		
$\bar{a}_3b_3$	$\bar{a}_2b_3$	$\bar{a}_1b_3$	$\bar{a}_0b_3$			
			b_3			
$x_3.$	x_2	x_1	x_0			

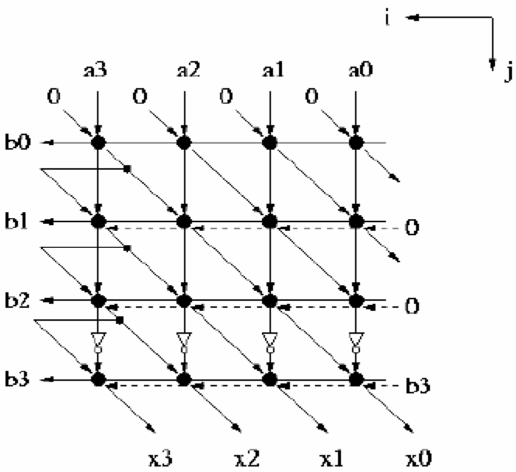
Not used if the result is truncated

Multiplication (Horners Rule):Parallel Carry Ripple Multiplier

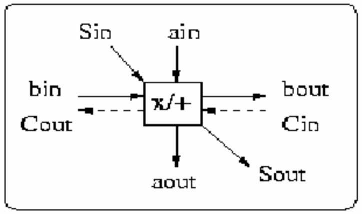
Negative MSBs solved with sign extension, one in each partial product



Not used if the result is truncated



Sign extension, one in each partial product

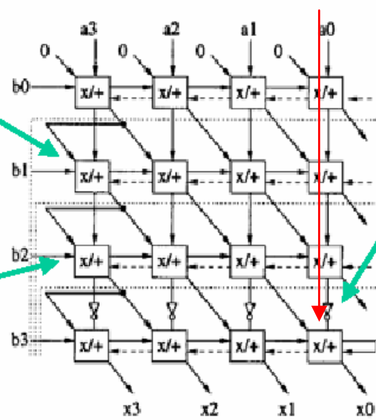


→ Broadcast signals:
bout=bin; aout=ain

→ Single-bit Full-Adder:
2 Cout + Sout = ain*bin + Sin + Cin

$$a_3b_0 + a_3b_1$$

$$+ a_3b_2$$



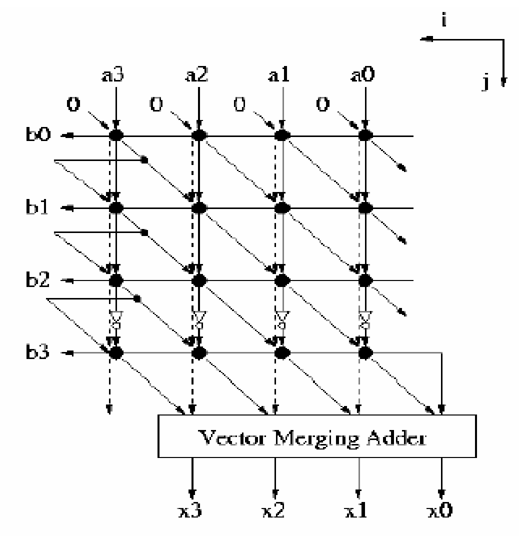
One's Complement

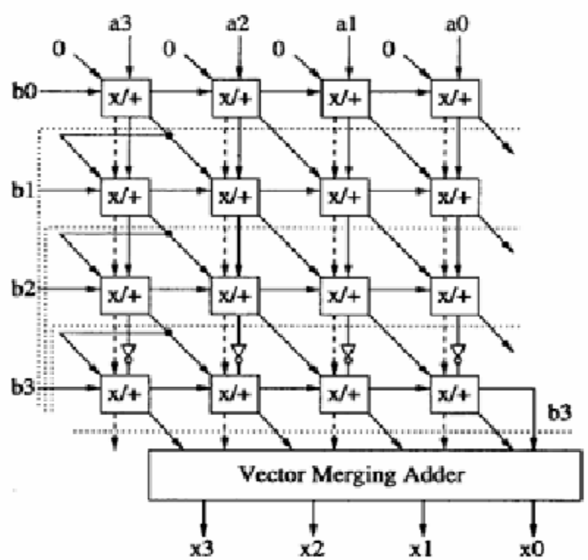
"LSB one"

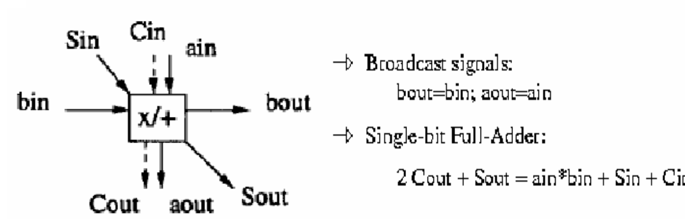
Good for layout!, But, the carry-propagation limits the speed of multiplication => carry-save adder (CSA)

Carry Save Array Multiplier

- CSA: The carries generated during the addition of a pair of operands can be saved and added, with proper alignment, to the next operands
- This leads to the concept of carry-save addition
- In the carry save array multiplier, the carry outputs are saved and used in the adder in the next row.
- In this case, the partial product is replaced by a partial sum and a partial carry, which are saved and passed on to the next row.
- This carry-save addition can be applied to all but the last step (partial sum and partial carry addition: vector merging adder (VMA, see p.486, Fig 13.7))
- Advantage of CSA: additions at different bit positions in the same row are independent of each other and can be carried out in parallel => speed up the multiplication







Modified Booth Algorithm

Recode binary numbers $x_i \in \{0,1\}$ **to** $y_i \in \{-2,-1,0,1,2\}$

- Five possible digits in y_i – radix 5?
- Overlapping method is used to reach radix 4
- Five digits require coding by 3 binary bits

  Two binary and one overlapping bit is used

Modified Booth Algorithm

- Example of the overlapping method
  - If X is 2's complement number

$$X = -x_{k-1} \times 2^{k-1} + \sum_{i=k-2}^{0} 2^i \times x_i \quad x_i \in \{0,1\}$$

Example $k = 6$

$$X = -32x_5 + 16x_4 + 8x_3 + 4x_2 + 2x_1 + x_0$$

$$X = 16(x_3 + x_4 - 2x_5) + 4(x_1 + x_2 - 2x_3) + (x_{-1} + x_0 - 2x_1)$$

Overlapping of x3, x1!

If $y_i = x_{i-1} + x_i - 2x_{i+1}$

$$X = Y = 16y_4 + 4y_2 + y_0 \quad y_i \in \{-2,-1,0,1,2\}$$

$$Y = \sum_{i=k-2}^{0} 2^i \times y_i \quad \text{n, i even} \Rightarrow Y = \sum_{i=\frac{k}{2}-1}^{0} 4^i \times y_{2i} \quad \text{i.e. Radix 4)}$$

Modified Booth Algorithm

$$y_i = -2x_{i+1} + x_i + x_{i-1}$$

x_{i+1}	x_i	x_{i-1}	y_i
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	-2
1	0	1	-1
1	1	0	-1
1	1	1	0

Examples:

$$X = 01\ \ 11\ \ 01\ \ 11\ \ (0) \Rightarrow Y = 02\ \ 0\bar{1}\ \ 02\ \ 0\bar{1}$$

$$X = 00\ \ 10\ \ 01\ \ 11\ \ (0) \Rightarrow Y = 01\ \ 0\bar{2}\ \ 00\ \ 0\bar{1}$$

$$X = 10\ \ 11\ \ 10\ \ 11\ \ (0) \Rightarrow Y = 0\bar{1}\ \ 00\ \ 0\bar{1}\ \ 0\bar{1}$$

There will always be at least one “0” in each pair

Modified Booth Algorithm

					0	1	0	1				5
x					0	1	1	1				7
					0	1	0	1		1	x	5
				0	1	0	1			2	x	5
			0	1	0	1				4	x	5
+		0	0	0	0					0	x	5
	0	0	1	0	0	0	1	1				

					0	1	0	1						5
x						2		-1						7
	1	1	1	1	1	0	1	1					-	5
+		0	1	0	1					2	x	4	x	5
	0	0	1	0	0	0	1	1						

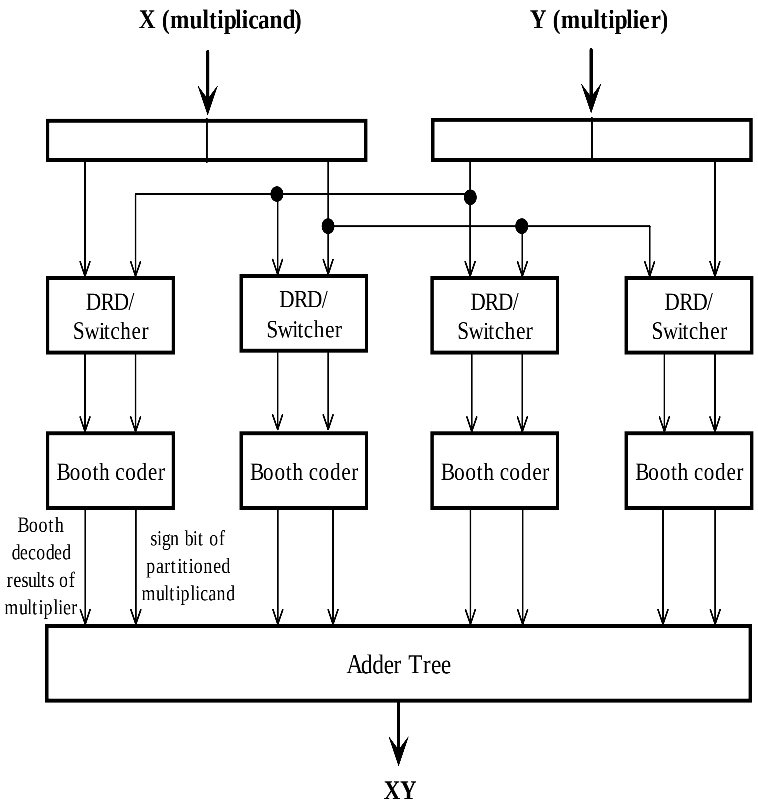
-1 ⇒ two´s complement conversion

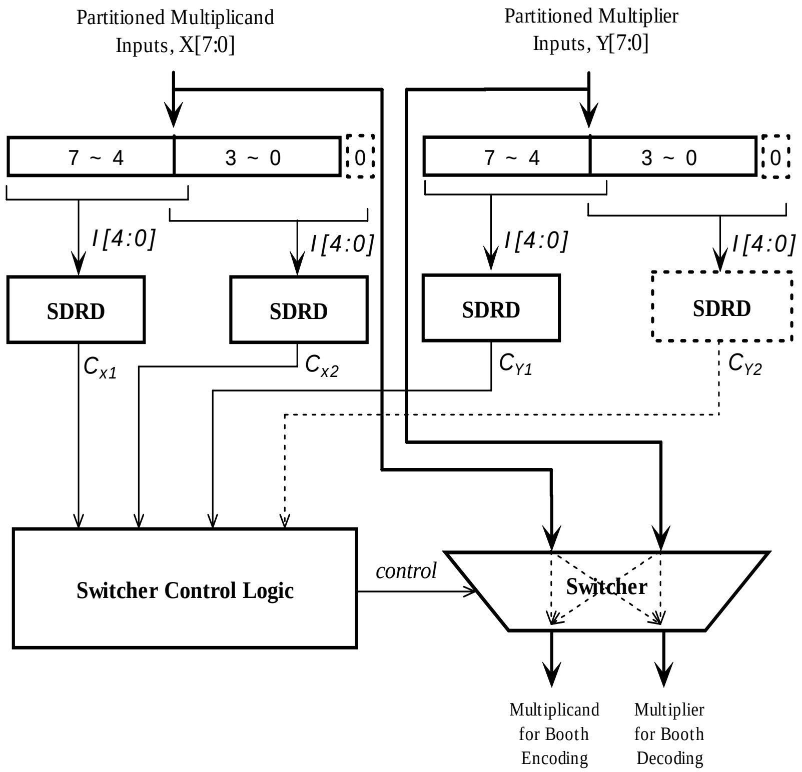
2 ⇒ shift one step (multiply by two)

-2 ⇒ two´s complement conversion and shift

Low-Power Modified Booth Algorithm

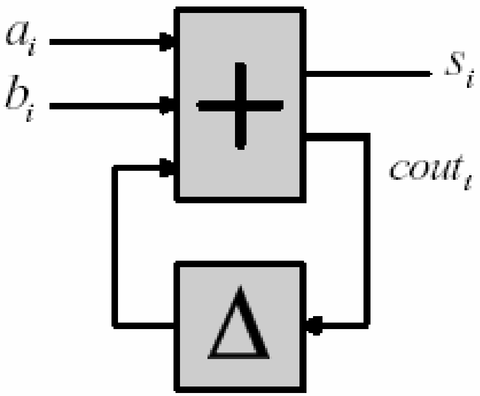
- Low-Power Booth Multiplier Proposed by KHU VLSI Lab: 20% power saving!





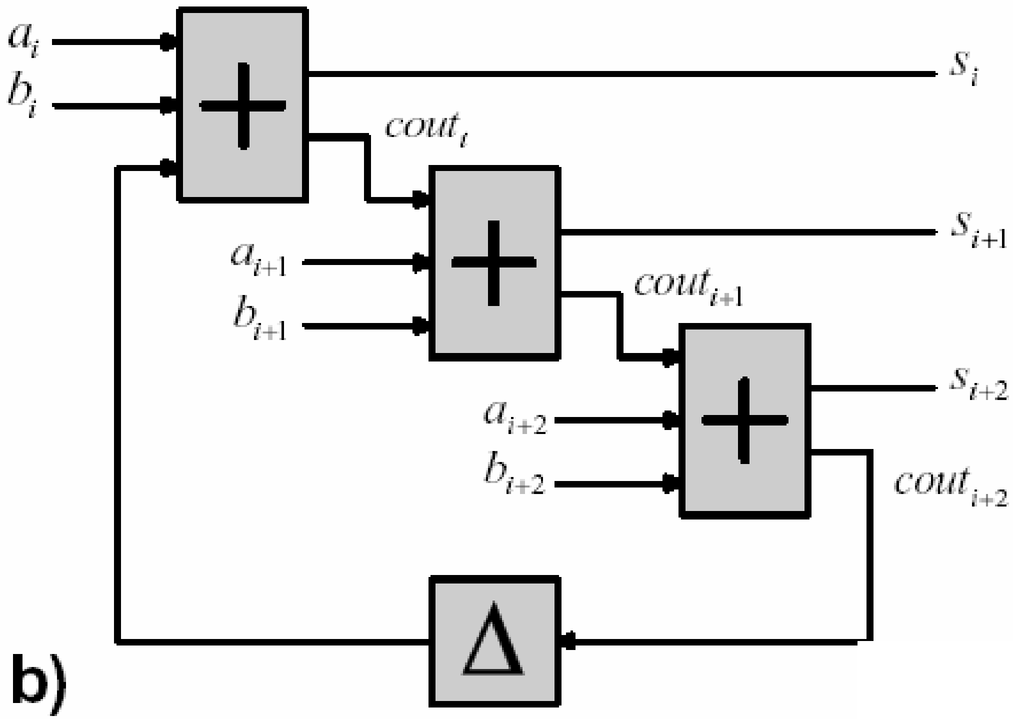
Serial Addition

Bit-Serial



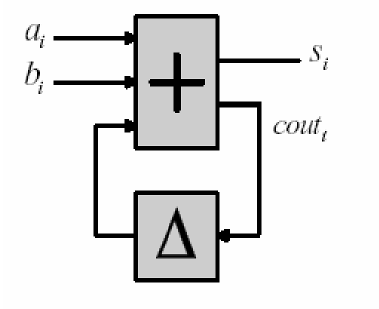
a)

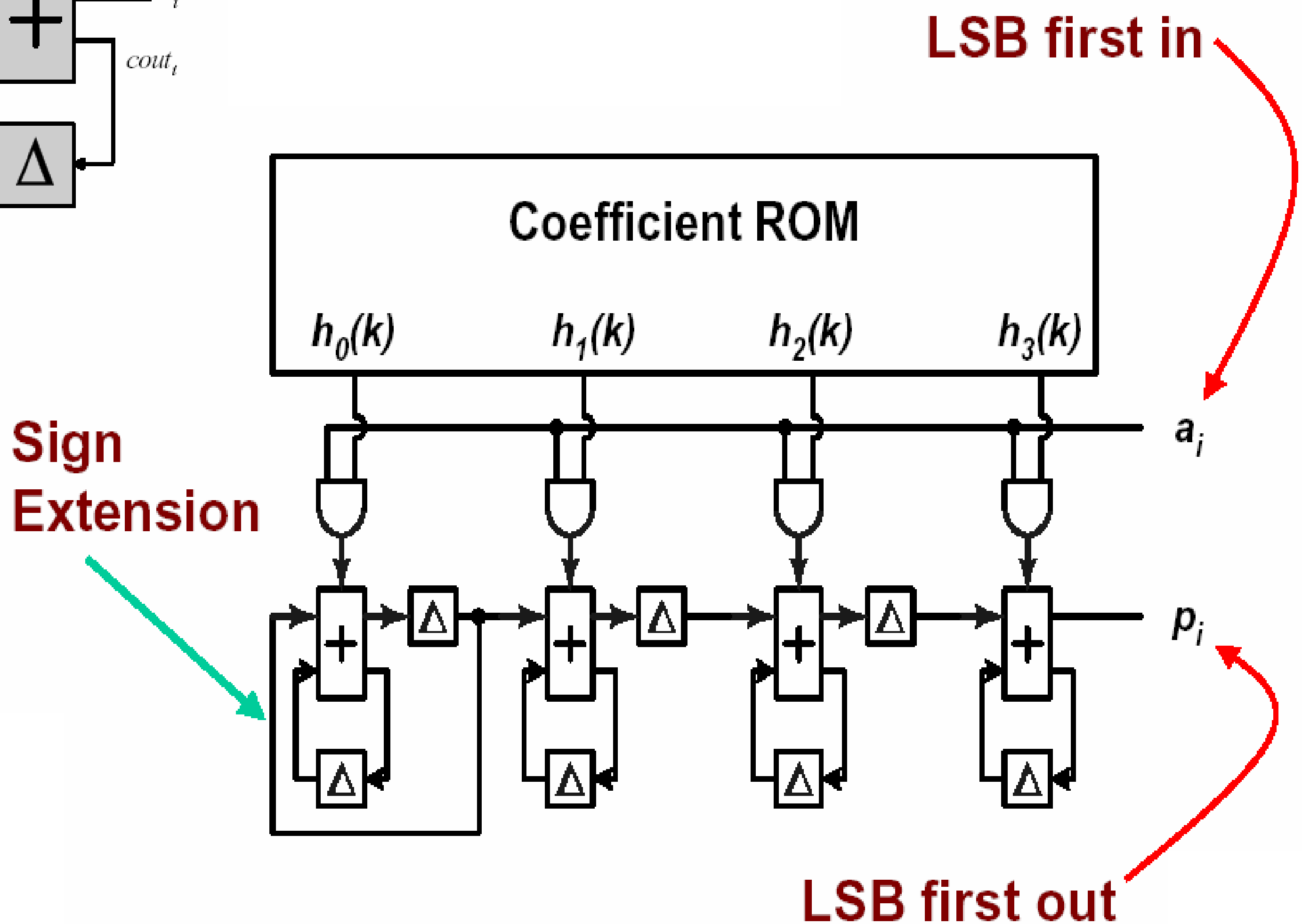
Digit-Serial



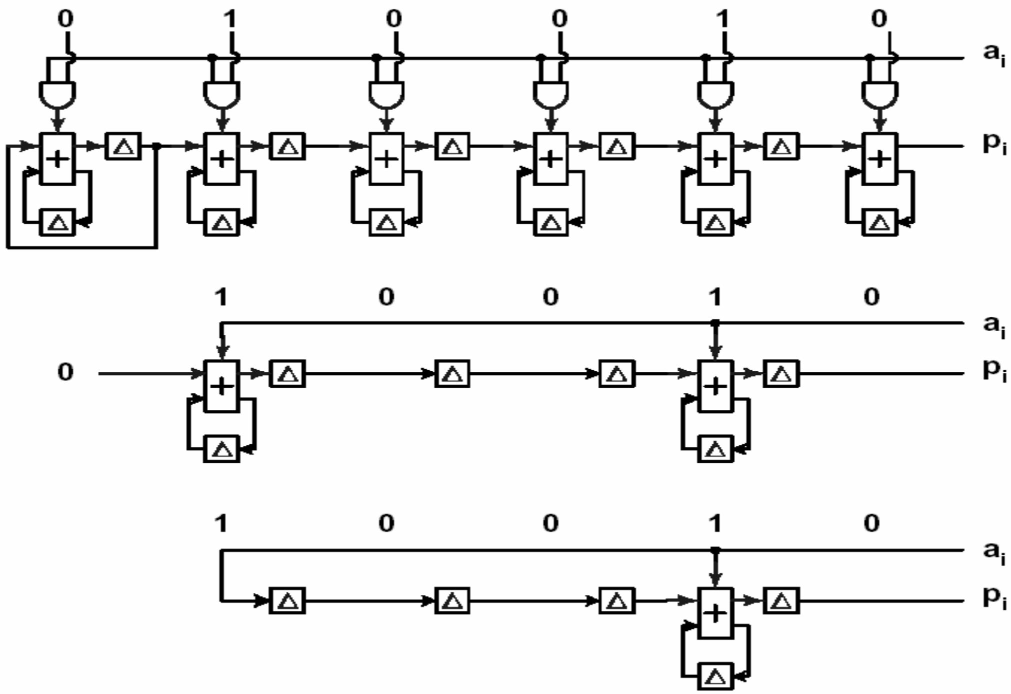
b)

Bit-Serial Multiplication





Fixed Coefficient Multiplication



Saves more than 1/2 of the adders at an average

- Fixed coefficient multiplication can be applied on array multipliers
  - Example: unused parts of the multiplier are removed in synthesized designs.

Signed Digit

A redundant representation where $x \in \{-1,0,1\}$

Example:

0 0 0 1 = 0 0 1 -1 = 0 1 -1 -1 ......

A sequence of ones:

0 1 1 1 1 0 = 1 0 0 0 -1 0

16 + 8 + 4 + 2 = 32 - 2

Canonical Signed Digit (CDS)

A sequence of ones can be replaced with:

1. A "-1" at the least significant position of the sequence.

2. A "1" at the position to the left of the most significant position of the sequence.

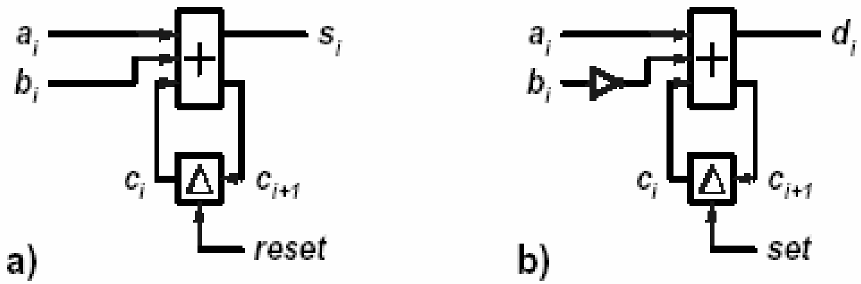
3. Zeros between the "1" and the "–1"

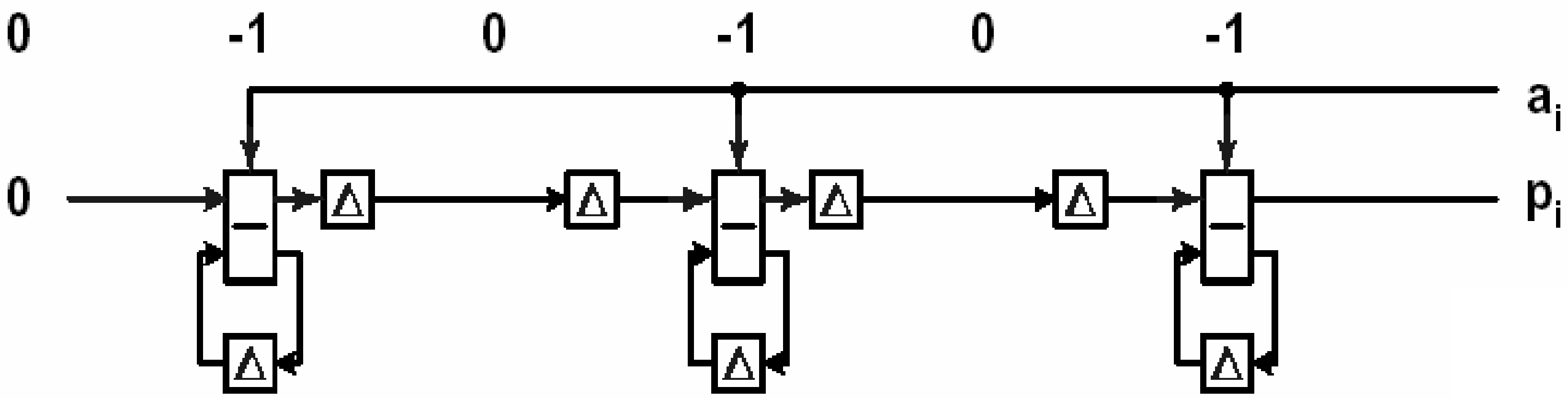
1	1	1	0	1	0	1	1
1	1	1	0	1	1	0	-1
1	1	1	1	0	-1	0	-1
0	0	0	-1	0	-1	0	-1

Saves more than 2/3 of the adder cells at an average

Canonical Signed Digit

1	1	1	0	1	0	1	1
1	1	1	0	1	1	0	-1
1	1	1	1	0	-1	0	-1
0	0	0	-1	0	-1	0	-1





Signed Digit Representation

- Modified Booth algorithm
  - Good for multiplication with variable coefficients
- Canonical signed digit
  - optimal for multiplication with fixed coefficients

Distributed Arithmetic

Often used in summation of inner products

See DCT in Chapter 9

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} c_2 & c_2 & c_2 & c_2 \\ c_1 & c_3 & -c_3 & -c_1 \\ c_2 & -c_2 & -c_2 & c_2 \\ c_3 & -c_1 & c_1 & -c_3 \end{bmatrix} \times \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$

Distributed Arithmetic

Sum of inner products

$$Y = \sum_{i=0}^{N-1} c_i x_i = c_0 x_0 + c_1 x_1 + c_2 x_2 \cdots$$

c_i are M-bit constants and x_i are W-bit numbers:

$$x_i = -x_{i,W-1} + \sum_{j=1}^{W-1} x_{i,W-1-j} \times 2^{-j}$$

$$Y = \sum_{i=0}^{N-1} c_i x_i = \sum_{i=0}^{N-1} c_i \overbrace{(-x_{i,W-1} + \sum_{j=1}^{W-1} x_{i,W-1-j} \times 2^{-j})}^{\text{Bits in the word}} =$$

$$= -\sum_{i=0}^{N-1} c_i x_{i,W-1} + \sum_{i=0}^{N-1} \left[\sum_{j=1}^{W-1} c_i x_{i,W-1-j} \times 2^{-j} \right] =$$

$$= -\sum_{i=0}^{N-1} c_i x_{i,W-1} + \sum_{j=1}^{W-1} \left[\sum_{i=0}^{N-1} c_i x_{i,W-1-j} \right] \times 2^{-j} =$$

Interchanged summation order

Same bit weight

Distributed Arithmetic

$$Y = c_0 x_0 + c_1 x_1 + c_2 x_2 =$$

Traditional summation order

$$-c_0 x_{0,2} + c_0 x_{0,1} 2^{-1} + c_0 x_{0,0} 2^{-2} +$$
$$-c_1 x_{1,2} + c_1 x_{1,1} 2^{-1} + c_1 x_{1,0} 2^{-2} +$$
$$-c_2 x_{2,2} + c_2 x_{2,1} 2^{-1} + c_2 x_{2,0} 2^{-2}$$

Note: c_i are M-bit constants and $x_{i,j}$ are single bits

$$Y = c_0 x_0 + c_1 x_1 + c_2 x_2 =$$

Interchanged summation order

Sign bits

$$\begin{array}{ccccc} -c_0 x_{0,2} & + & c_0 x_{0,1} 2^{-1} & + & c_0 x_{0,0} 2^{-2} + \\ -c_1 x_{1,2} & + & c_1 x_{1,1} 2^{-1} & + & c_1 x_{1,0} 2^{-2} + \\ -c_2 x_{2,2} & + & c_2 x_{2,1} 2^{-1} & + & c_2 x_{2,0} 2^{-2} \end{array}$$

Distributed Arithmetic

Interchanged summation order (rewritten)

Sign bits

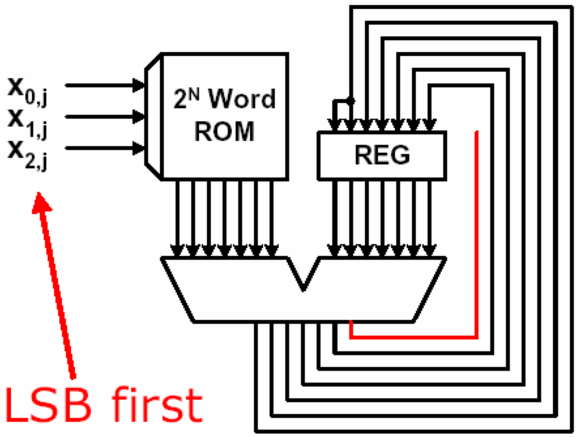
$$-(c_0 x_{0,2} + c_1 x_{1,2} + c_2 x_{2,2}) +$$

$$+(c_0 x_{0,1} + c_1 x_{1,1} + c_2 x_{2,1}) \times 2^{-1} +$$

$$+(c_0 x_{0,0} + c_1 x_{1,0} + c_2 x_{2,0}) \times 2^{-2}$$

$x_{0,j}$	$x_{1,j}$	$x_{2,j}$	ROM
0	0	0	0
0	0	1	c_2
0	1	0	c_1
0	1	1	c_1+c_2
1	0	0	c_0
1	0	1	c_0+c_2
1	1	0	c_0+c_1
1	1	1	$c_0+c_1+c_2$

Shift Accumulator



LSB first
(from each sample)

$x_{0,j}$	$x_{1,j}$	$x_{2,j}$	ROM
0	0	0	0
0	0	1	c_2
0	1	0	c_1
0	1	1	c_1+c_2
1	0	0	c_0
1	0	1	c_0+c_2
1	1	0	c_0+c_1
1	1	1	$c_0+c_1+c_2$

Distributed Arithmetic: Example

$x_{0,j} = 0.11$ $c_0 = 0.00$

$x_{1,j} = 0.10$ $c_1 = 0.01$

$x_{2,j} = 0.01$ $c_2 = 0.10$

$x_{0,j}$	$x_{1,j}$	$x_{2,j}$	ROM	Coeff.
0	0	0	0.00	0
0	0	1	0.10	c_2
0	1	0	0.01	c_1
0	1	1	0.11	c_1+c_2
1	0	0	0.00	c_0
1	0	1	0.10	c_0+c_2
1	1	0	0.01	c_0+c_1
1	1	1	0.11	$c_0+c_1+c_2$

$$Sum = rom_0 + \frac{1}{2} rom_6 + \frac{1}{4} rom_5 =$$

$$= 0.00 + 0.001 + 0.0010 = 0.0100$$

Newton Raphson

$$f(x) = \frac{1}{x} - d; \;\; f'(x) = -\frac{1}{x^2}$$

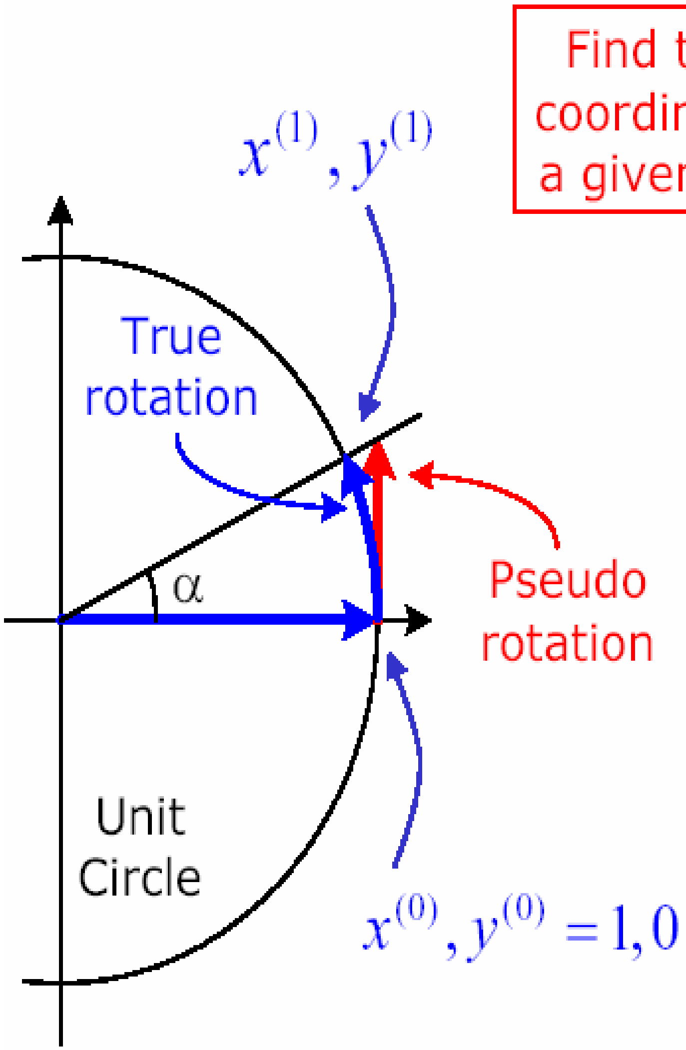
$$x(i+1) = x(i) - \frac{f(x(i))}{f'(x(i))} = x(i) - \frac{\frac{1}{x(i)} - d}{-\frac{1}{x^2(i)}} = x(i) + \frac{x^2(i)}{x(i)} - dx^2(i)$$

$$x(i+1) = 2x(i) - dx^2(i)$$

CORDIC Algorithm

- Many applications
  - Polar/rectangular conversion
  - Sin, cos, tan, ...
  - Arcsin, arccos, arctan, ...
  - Hyperbolic functions
  - Division
  - Square-root
- No multiplications is needed
- One bit accuracy per iteration

CORDIC Algorithm: Real Rotation



$$x^{(i+1)} = x^{(i)} \cos\alpha^{(i)} - y^{(i)} \sin\alpha^{(i)}$$

$$x^{(i+1)} = \frac{x^{(i)} - y^{(i)} \tan\alpha^{(i)}}{\sqrt{1+\tan^2\alpha^{(i)}}}$$

$$y^{(i+1)} = y^{(i)} \cos\alpha^{(i)} + x^{(i)} \sin\alpha^{(i)}$$

$$y^{(i+1)} = \frac{y^{(i)} + x^{(i)} \tan\alpha^{(i)}}{\sqrt{1+\tan^2\alpha^{(i)}}}$$

Example:

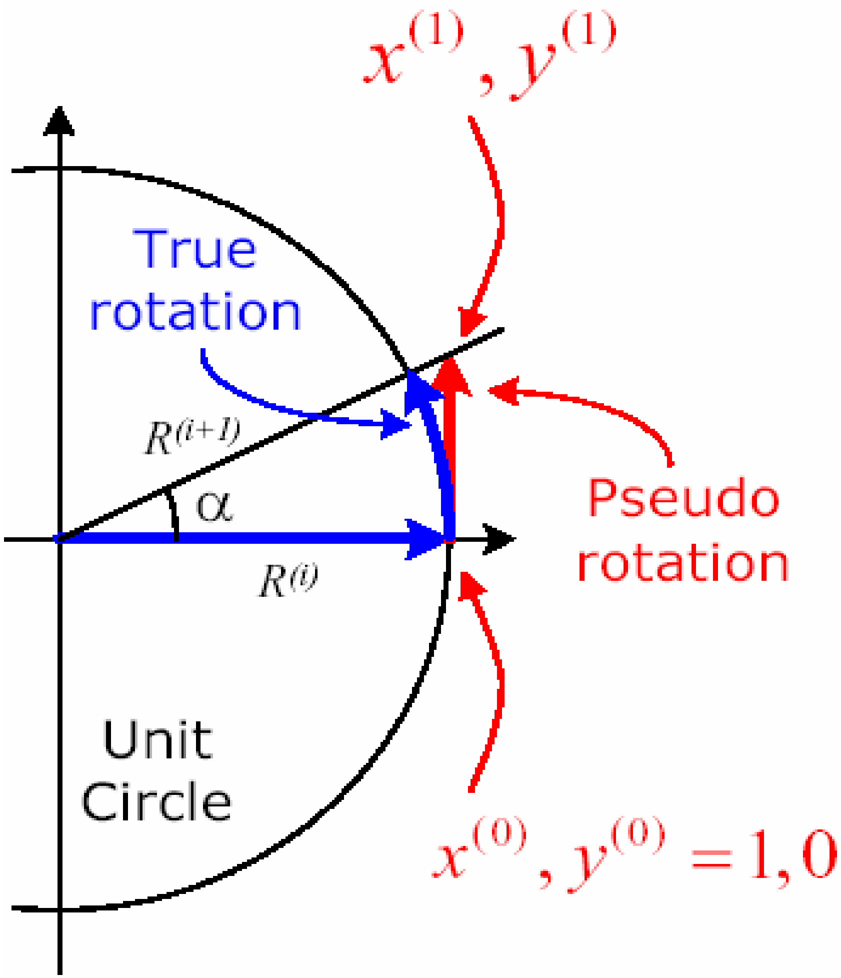
$$x^{(1)} = \frac{x^{(0)} - y^{(0)} \tan\alpha^{(i)}}{\sqrt{1+\tan^2\alpha^{(i)}}}$$

$$x^{(1)} = \frac{1}{\sqrt{1+\tan^2\alpha^{(i)}}}$$

$$y^{(1)} = \frac{y^{(0)} - x^{(0)} \tan\alpha^{(i)}}{\sqrt{1+\tan^2\alpha^{(i)}}}$$

$$y^{(1)} = \frac{\tan\alpha^{(i)}}{\sqrt{1+\tan^2\alpha^{(i)}}}$$

CORDIC Algorithm: Pseudo Rotation



$$x^{(i+1)} = x^{(i)} - y^{(i)} \tan \alpha^{(i)}$$

$$y^{(i+1)} = y^{(i)} + x^{(i)} \tan \alpha^{(i)}$$

Example:

$$x^{(1)} = x^{(0)} - y^{(0)} \tan \alpha^{(i)} = 1$$

$$y^{(1)} = y^{(0)} + x^{(0)} \tan \alpha^{(i)} = \tan \alpha^{(i)}$$

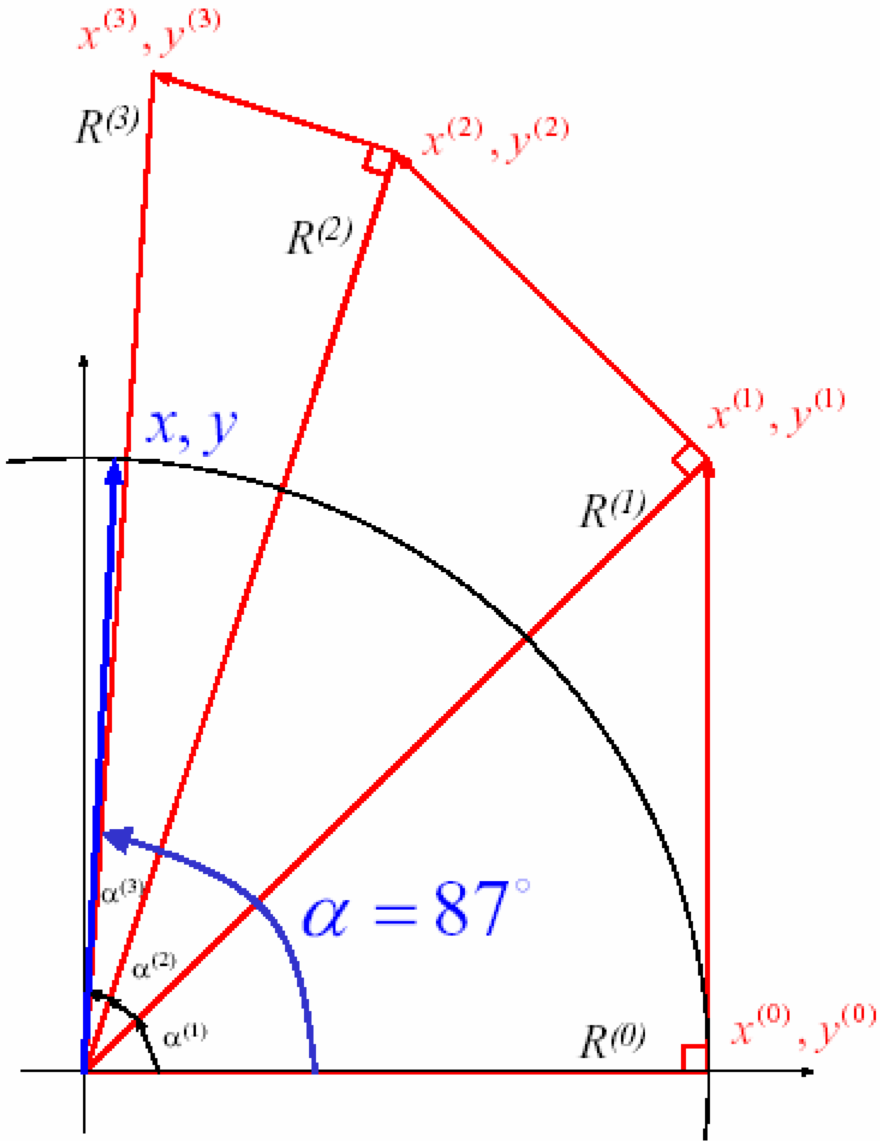
However the length $R > 1$

$$R^{(i+1)} = R^{(i)} \frac{1}{\cos \alpha^{(i)}} =$$

$$= \left\{ \frac{1}{\cos^2 \alpha^{(i)}} = 1 + \tan^2 \alpha^{(i)} \right\} =$$

$$R^{(i+1)} = R^{(i)} \sqrt{1 + \tan^2 \alpha^{(i)}}$$

CORDIC Algorithm: Pseudo Rotation



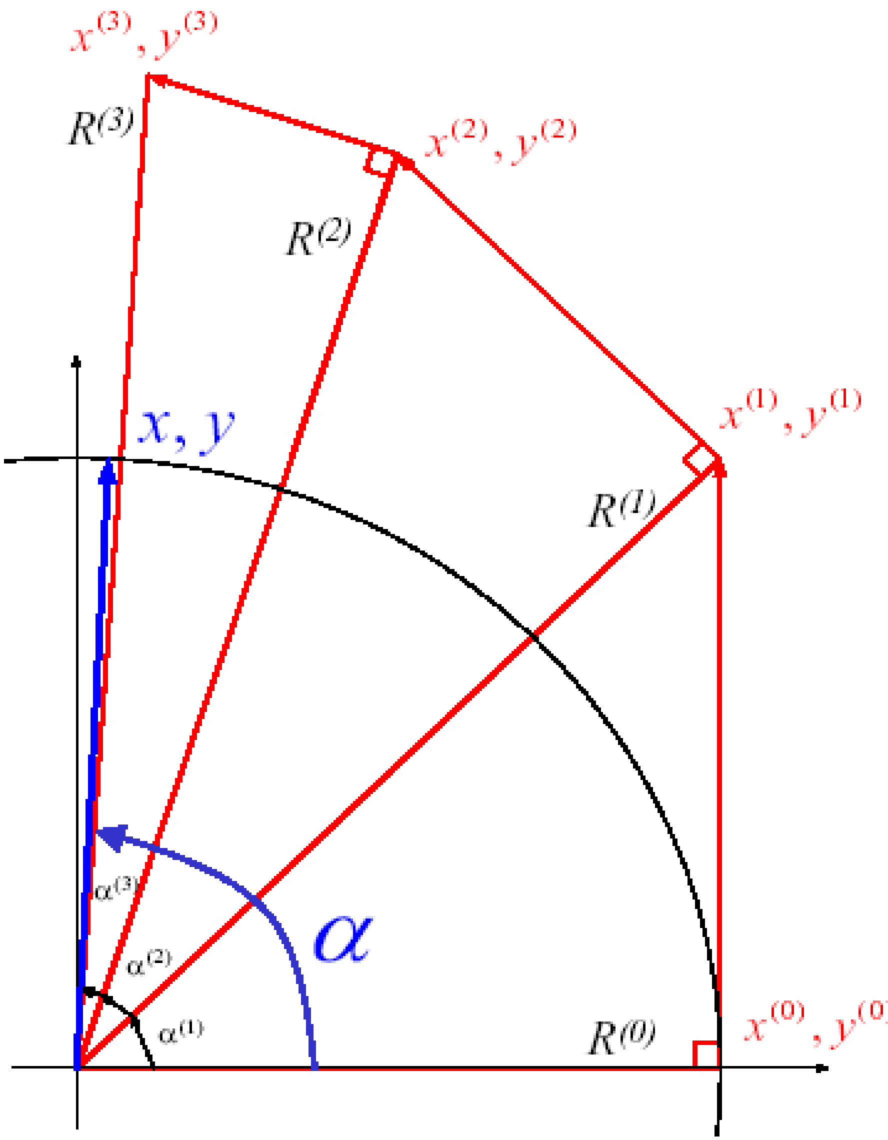
The Angle α is known

Derive *x,y* using
three iterations where

$$\alpha - \alpha^{(1)} - \alpha^{(2)} - \alpha^{(3)} \rightarrow 0$$

$$87 - 45.0 - 26.6 - 14.0 = 1.4^\circ$$

CORDIC Algorithm: Three Iterations



The vector length R is increasing durig each iterations

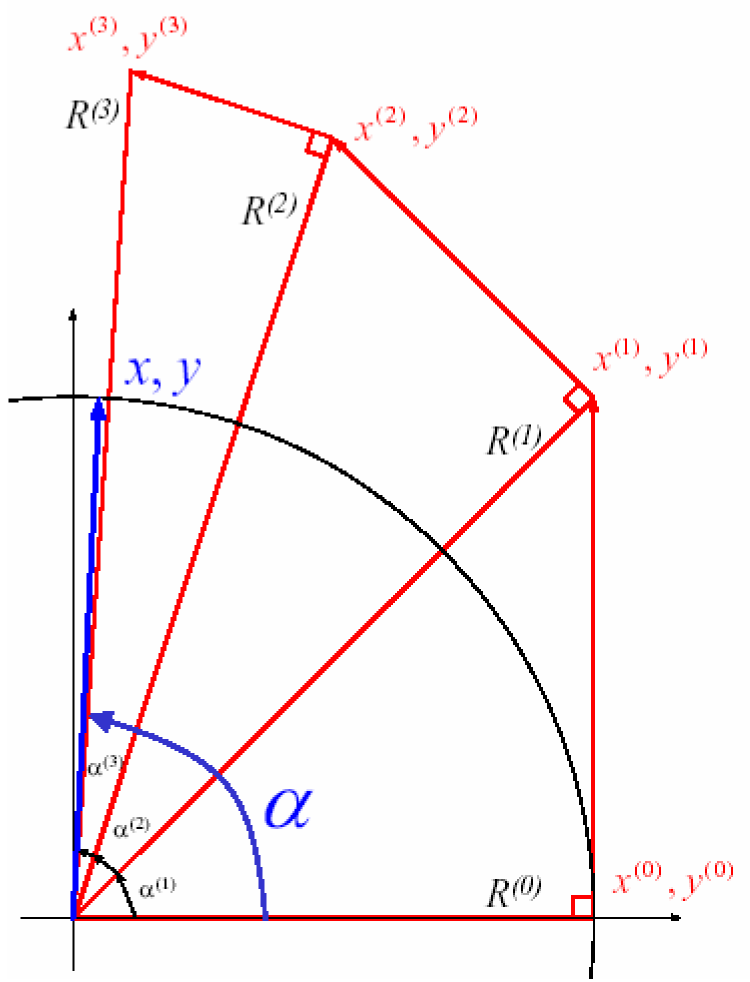
$$R^{(0)} = 1$$

$$R^{(1)} = R^{(0)}\sqrt{1+\tan^2\alpha^{(1)}} = \sqrt{1+\tan^2 45^\circ} = \sqrt{2} = 1.41$$

$$R^{(2)} = R^{(1)}\sqrt{1+\tan^2\alpha^{(2)}} = \sqrt{2}\sqrt{1+\tan^2 26.6^\circ} = \sqrt{\frac{5}{2}} = 1.58$$

$$R^{(3)} = R^{(2)}\sqrt{1+\tan^2\alpha^{(3)}} = \sqrt{\frac{5}{2}}\sqrt{1+\tan^2 14.0^\circ} = \sqrt{\frac{85}{32}} = 1.63$$

CORDIC Algorithm



Derive $x^{(3)}, y^{(3)}$

$$\tan\alpha^{(1)} = 1; \quad \tan\alpha^{(2)} = \frac{1}{2}; \quad \tan\alpha^{(3)} = \frac{1}{4}$$

$$x^{(i+1)} = x^{(i)} - y^{(i)} \tan\alpha^{(i)}$$

$$y^{(i+1)} = y^{(i)} + x^{(i)} \tan\alpha^{(i)}$$

$$\begin{cases} x^{(1)} = x^{(0)} - y^{(0)} \times 1 = 1 \\ y^{(1)} = y^{(0)} + x^{(0)} \times 1 = 1 \end{cases}$$

$$\begin{cases} x^{(2)} = x^{(1)} - y^{(1)} \times \frac{1}{2} = \frac{1}{2} \\ y^{(2)} = y^{(1)} + x^{(1)} \times \frac{1}{2} = \frac{3}{2} \end{cases}$$

$$\begin{cases} x^{(3)} = x^{(2)} - y^{(2)} \times \frac{1}{4} = \frac{1}{8} \\ y^{(3)} = y^{(2)} + x^{(2)} \times \frac{1}{4} = \frac{13}{8} \end{cases}$$

CORDIC Algorithm

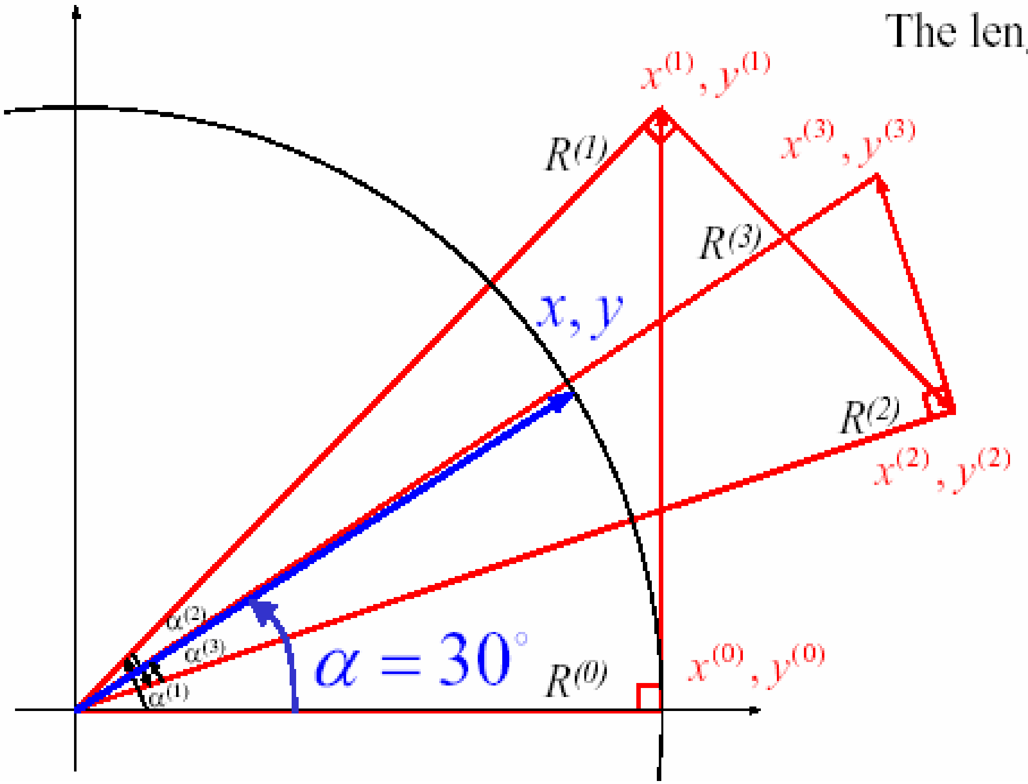
$$x, y \approx \frac{x^{(3)}}{R^{(3)}}, \frac{y^{(3)}}{R^{(3)}}$$

$$\alpha = 30^\circ \quad \Rightarrow \quad \text{Pos. Rot.}$$

$$\alpha - \alpha^{(1)} = 30 - 45 = -15^\circ \quad \Rightarrow \quad \text{Neg. Rot.}$$

$$\alpha - \alpha^{(1)} - \alpha^{(2)} = -15 + 26.6 = 11.6^\circ \quad \Rightarrow \quad \text{Pos. Rot.}$$

$$\alpha - \alpha^{(1)} - \alpha^{(2)} - \alpha^{(3)} = 11.6 - 14 = 2.4^\circ \quad \Rightarrow \quad \text{Neg. Rot.}$$



The lengths $R^{(i)}$ are constant (precalculated)

$$R^{(0)} = 1$$

$$R^{(1)} = \sqrt{2}$$

$$R^{(2)} = \sqrt{\frac{5}{2}}$$

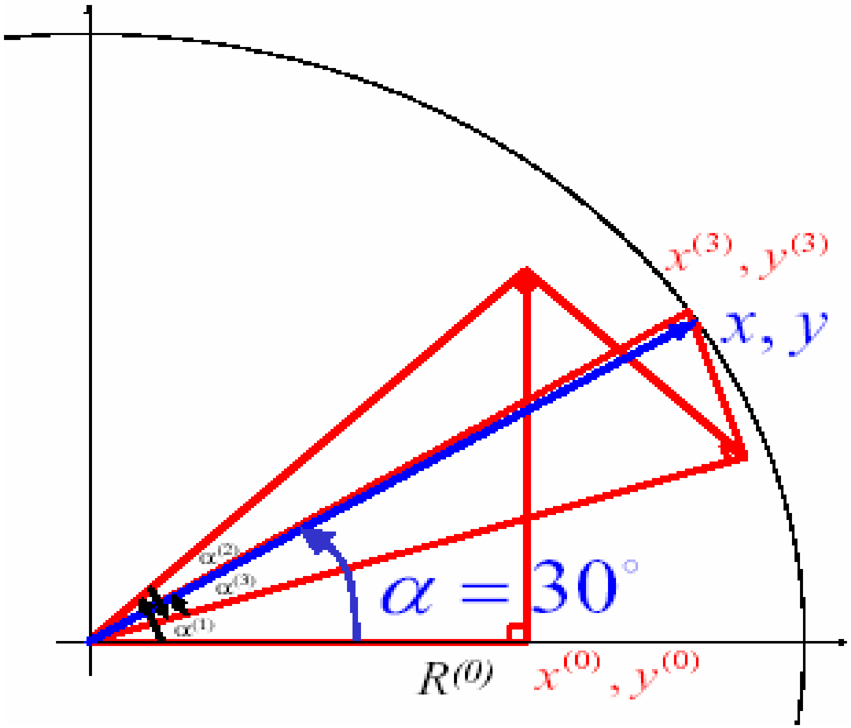
$$R^{(3)} = \sqrt{\frac{85}{32}}$$

CORDIC Algorithm

Start at $(x^{(0)}, y^{(0)}) =$

$$= \left(\frac{1}{R^{(3)}}, 0\right) =$$

$$= \left(\sqrt{\frac{32}{85}}, 0\right)$$



Derive new coordinats

$$(x, y) \approx (x^{(3)}, y^{(3)})$$

Derive $\cos\alpha$ and $\sin\alpha$

$$x^{(3)} = R^{(3)}\left[x^{(0)} \cos\sum\alpha^{(i)} - y^{(0)} \sin\sum\alpha^{(i)}\right] =$$

$$= \cos\sum\alpha^{(i)} \approx \cos\alpha$$

$$y^{(3)} = R^{(3)}\left[y^{(0)} \cos\sum\alpha^{(i)} + x^{(0)} \sin\sum\alpha^{(i)}\right]$$

$$= \sin\sum\alpha^{(i)} \approx \sin\alpha$$

Derive $\tan\alpha$

$$\tan\alpha^{(i)} = \frac{\sin\sum\alpha^{(i)}}{\cos\sum\alpha^{(i)}} \approx \tan\alpha; \quad \text{(division needed)}$$

Basic CORDIC Rotations

Shifts

Angles Prestored

0	$\tan\alpha^{(0)} = 0$	$\alpha^{(0)} = \arctan 0 = 0$
1	$\tan\alpha^{(1)} = 1$	$\alpha^{(0)} = \arctan 1 = 45^\circ$
2	$\tan\alpha^{(2)} = \frac{1}{2}$	$\alpha^{(0)} = \arctan\frac{1}{2} = 26.6^\circ$
3	$\tan\alpha^{(3)} = \frac{1}{4}$	$\alpha^{(0)} = \arctan\frac{1}{4} = 14.0^\circ$
4	$\tan\alpha^{(4)} = \frac{1}{8}$	$\alpha^{(0)} = \arctan\frac{1}{8} = 7.1^\circ$
5	$\tan\alpha^{(5)} = \frac{1}{16}$	$\alpha^{(0)} = \arctan\frac{1}{16} = 3.6^\circ$

Basic CORDIC Rotations

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

$$y^{(i+1)} = y^{(i)} + d_i \, x^{(i)} \frac{1}{2^i}$$

$$\alpha^{(i+1)} = \alpha^{(i)} - d_i \arctan \frac{1}{2^i}$$

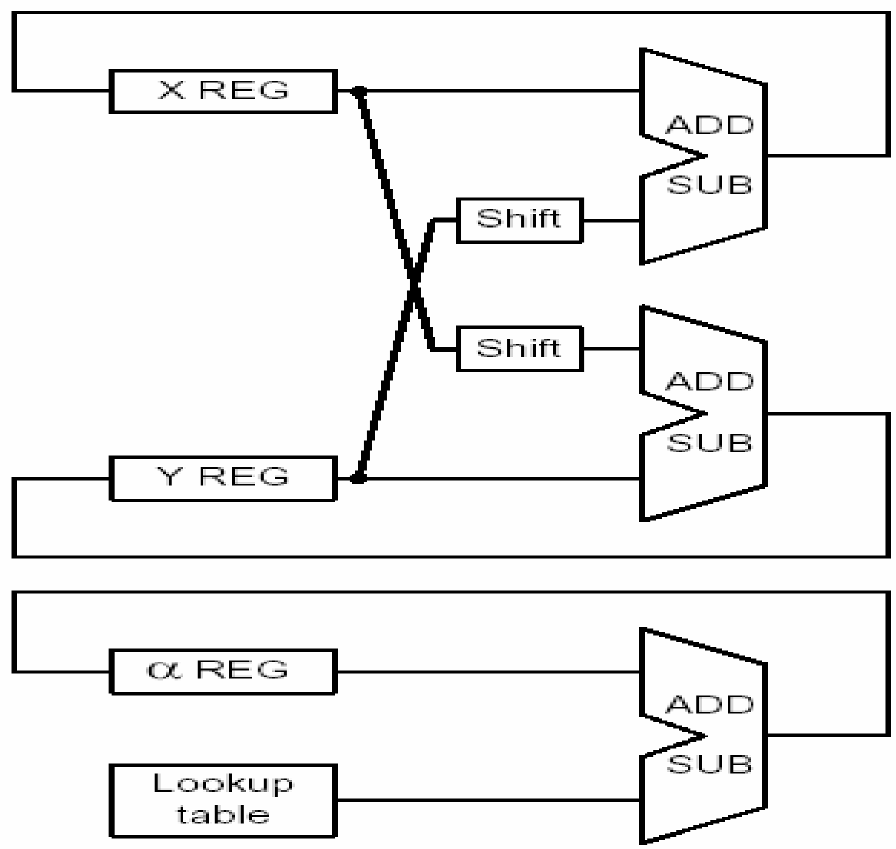
$$d_i = \mathrm{sign}(\alpha^{(i)})$$

Each CORDIC iteration require

- 3 ADD/SUB
- 2 Shifts

CORDIC Hardware

Each CORDIC iteration require
- 3 ADD/SUB
- 2 Shifts



Summary and Problems

- Summary

- Problems
  - ***No problem!***