DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Implementation of DSP IC

Lecture 9 Bit-Level Arithmetic Architecture

Fixed Point Representation

- A W-bit fixed point 2's complement number A is represented as:

$$A=a_{W-1}\circ a_{W-2}\ldots a_1 a_0$$

where the bits a_i, $0\le i\le W-1$, are either 0 or 1, and the msb is the sign bit.

- The value of A is in the range of $[-1,1-2^{-W+1}]$ and is given by

$$A=-a_{W-1}+\sum a_{W-1-i}2^{-i}$$

- For bit-serial implementations, constant word length multipliers are considered. For a W×W bit multiplication, the W most-significant bits of the (2W-1)-bit product are retained. (the others are discarded)

Fixed Point Multiplication

Let the multiplicand and the multiplier be A and B:

$$A = a_{W-1}.a_{W-2}\ldots a_1.a_0 = -a_{W-1} + \sum_{i=1}^{W-1} a_{W-1-i}2^{-i}$$

$$B = b_{W-1}.b_{W-2}\ldots b_1.b_0 = -b_{W-1} + \sum_{i=1}^{W-1} b_{W-1-i}2^{-i}$$

Their product is given by :

Full-precision

$$P = -p_{2W-2} + \sum_{i=1}^{2W-2} p_{2W-2-i}2^{-i}$$

The radix point is to the right of the msb p_{2W-2}

In constant word length multiplication, W – 1 lower order bits in the product P are ignored and the Product is denoted as $X \Leftarrow P = A \times B$, where

$$X = -x_{W-1} + \sum_{i=1}^{W-1} x_{W-1-i}2^{-i}$$

A constant word-length representation

Example: W=4

- $A = a_3 \circ a_2 a_1 a_0 = -a_3 + a_2 2^{-1} + a_1 2^{-2} + a_0 2^{-3}$
- $B = b_3 \circ b_2 b_1 b_0 = -b_3 + b_2 2^{-1} + b_1 2^{-2} + b_0 2^{-3}$

- Multiplication of $A \times B$

Using Horner's rule, multiplication of A and B can be written as

$$P = A \times (-b_{W-1} + \Sigma\, b_{W-1-i} 2^{-i})$$
$$= \underline{-A.\ b_{W-1}} + [A.\ b_{W-2} + [A.\ b_{W-3} + [\ldots + [A.\ \ b_1 + A\ b_0\ 2^{-1}]\ 2^{-1}]\ldots]2^{-1}]\ 2^{-1}$$

where 2^{-1} denotes scaling operation.

$A \times B$ with W=4

Horner's rule

			a_3	a_2	a_1	a_0	
			b_3	b_2	b_1	b_0	
			$-b_0a_3$	b_0a_2	b_0a_1	b_0a_0	→ b_0A
		$-b_1a_3$	b_1a_2	b_1a_1	b_1a_0		→ b_1A
	$-b_2a_3$	b_2a_2	b_2a_1	b_2a_0			→ b_2A
$-b_3\bar{a}_3$	$b_3\bar{a}_2$	$b_3\bar{a}_1$	$b_3\bar{a}_0$	→ $-b_3A$			
			b_3				
p_6	p_5	p_4	p_3	p_2	p_1	p_0	

The additions cannot be carried out directly due to the terms with negative weight !!

Sign Extension & Shift

$$A = a_3 \circ a_2 a_1 a_0 = -a_3 + a_2 2^{-1} + a_1 2^{-2} + a_0 2^{-3}$$
$$= \left(-a_3 2 + a_3\right) + a_2 2^{-1} + a_1 2^{-2} + a_0 2^{-3}$$
$$= a_3 a_3 \circ a_2 a_1 a_0 = \cdots$$

$$A \times 2^{-1} = \left(a_3 \circ a_2 a_1 a_0\right) 2^{-1} = \left(-a_3 + a_2 2^{-1} + a_1 2^{-2} + a_0 2^{-3}\right) 2^{-1}$$
$$= \left[\left(-a_3 2 + a_3\right) + a_2 2^{-1} + a_1 2^{-2} + a_0 2^{-3}\right] 2^{-1}$$
$$= -a_3 + a_3 2^{-1} + a_2 2^{-2} + a_1 2^{-3} + a_0 2^{-4}$$
$$= \underline{a_3 \circ a_3 a_2 a_1 a_0} \Rightarrow a_3 \circ a_3 a_2 a_1$$

The extension sign bit remains the msb position, while a_0 is eliminated !!

A×B with W=4

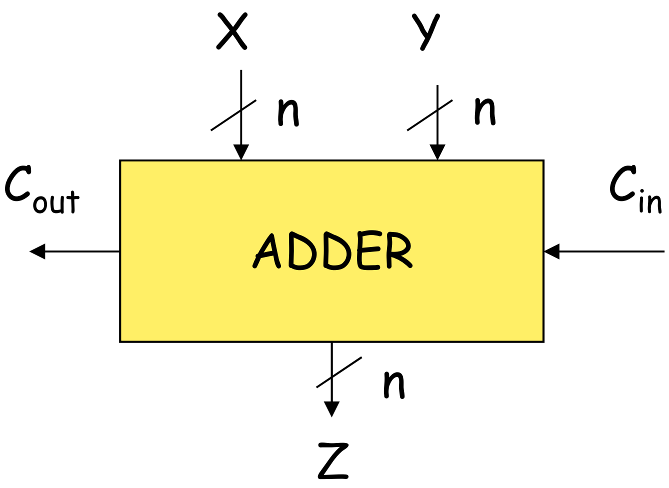
Negative MSBs solved with sign extension, one in each partial product

			$a_3.$	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

2's Complement System

- 2's complement adder



	2's complement	integer
$X = 1011$	$x = -5$	$xi = 11$
$Y = 0101$	$y = 5$	$yi = 5$
$W = 100000$		$w = 16$
$Z = 0000$	$z = 0$	

$x+y+c_{in}=2^n c_{out}+z$

To Speed Up Multiplication

- To accelerate accumulation
- To reduce the number of partial products

xxxx	1	0	0	0	0	0	*xxxxxx*
xxxx −	0	0	0	0	0	1	*xxxxxx*
xxxx		1	1	1	1	1	*xxxxxx*

To record into signed-bit representation with fewer number of nonzero bits

5 partial products may be required

Booth's Modified Algorithm

- Recode binary numbers $x_i \in \{0,1\}$ to $y_i \in \{-2,-1,0,1,2\}$ (5-level Booth recoding)
- Five possible digits in y_i radix-5 ?
- Overlapping method is used to reach radix 4
- Five digits require coding by 3 binary bits, that is two binary and one overlapping bit is used.
- The digits x_{i+1} and x_i are recoded into signed digit y_i, with x_{i-1} serving as a reference bit.

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

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:

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

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

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

There will always be at least one "0" in each pair

Example

					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						

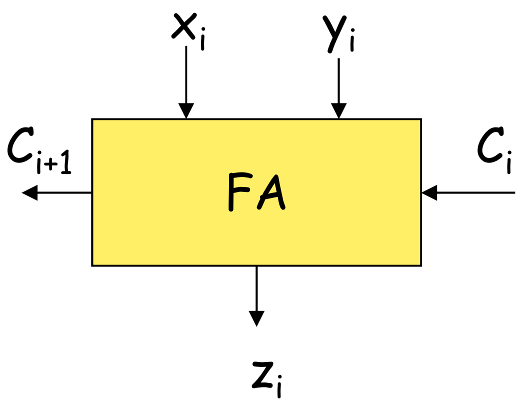
-1 → 2's complement conversion

2 → shift one step (or multiply by 2)

-2 → 2's complement conversion and shift

2-Operand Addition

- 1-bit adder (full-adder module)



$$z_i = (x_i + y_i + c_i) \bmod 2$$

$$c_{i+1} = \left\lfloor \frac{x_i + y_i + c_i}{2} \right\rfloor$$

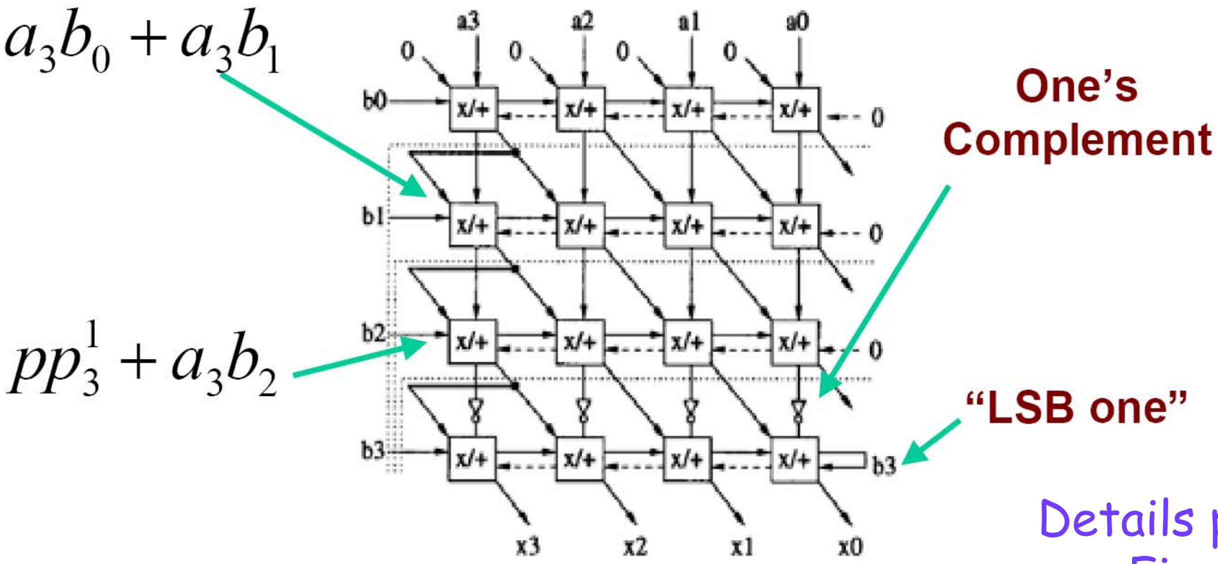
Adder schemes:

- Switched carry-ripple adder
- Carry-skip adder
- Carry-lookahead adder
- Prefix adder
- Carry-select adder
- Conditional-sum adder

- Carry-save adder
- Signed-digit adder

Parallel Carry-Ripple Array (CRA) Multiplier

- The carry output of an adder is rippled to the adder input to the left in the same row.
- Bit-level dependence graph (DG) of 4×4-bit carry-ripple array multiplication:



Details please see Fig. 13.4

Interleaved Approach for FIR

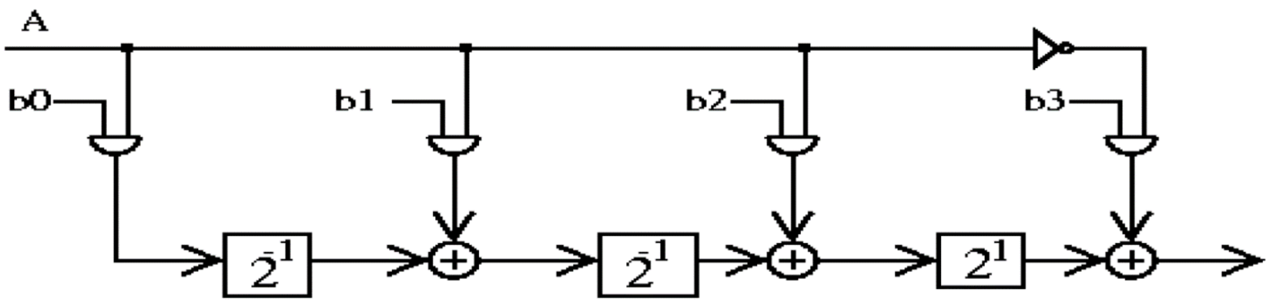
- $Y(n)=x(n) + \underline{f\ x(n-1)} + \underline{g\ x(n-2)}$, f and g are constants
- To perform the computation and accumulation of partial products associated with f and g simultaneously
- Since the truncation is applied at the final step, it leads to better accuracy.
- If the coefficients are interleaved in such a way that their partial products are computed in different rows, the resulting architecture is called bit-plane architecture

Bit-serial 2's Complement Multiplier

Using Horner's rule, multiplication of A and B can be written as

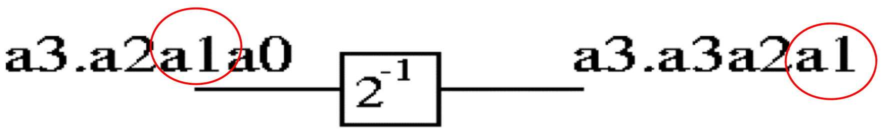
$$P = A \times (-b_{W-1} + \Sigma\, b_{W-1-i} 2^{-i})$$
$$= -A.\, b_{W-1} + [A.\, b_{W-2} + [A.\, b_{W-3} + [\ldots + [A.\ \ b_1 + A\, b_0\, 2^{-1}]\, 2^{-1}]\ldots]2^{-1}]\, 2^{-1}$$

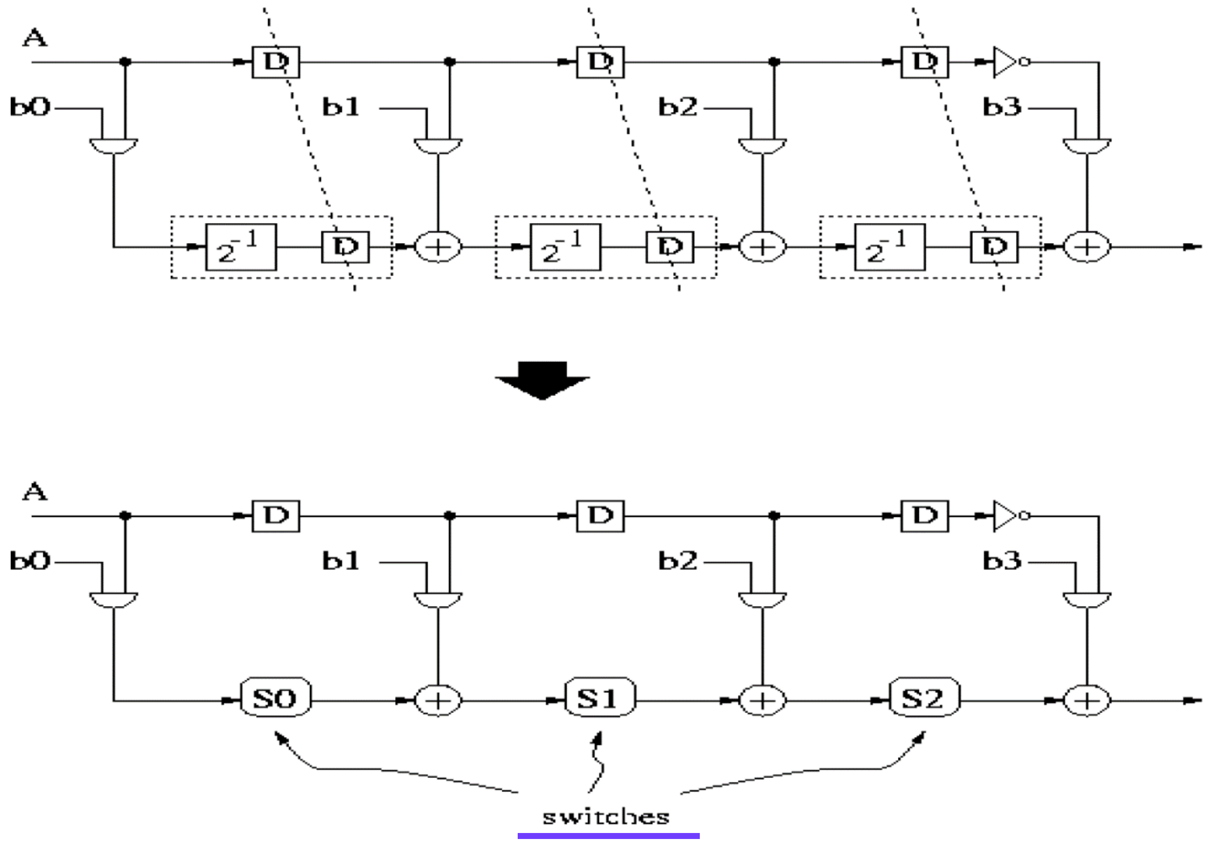
where 2^{-1} denotes scaling operation.



(a)

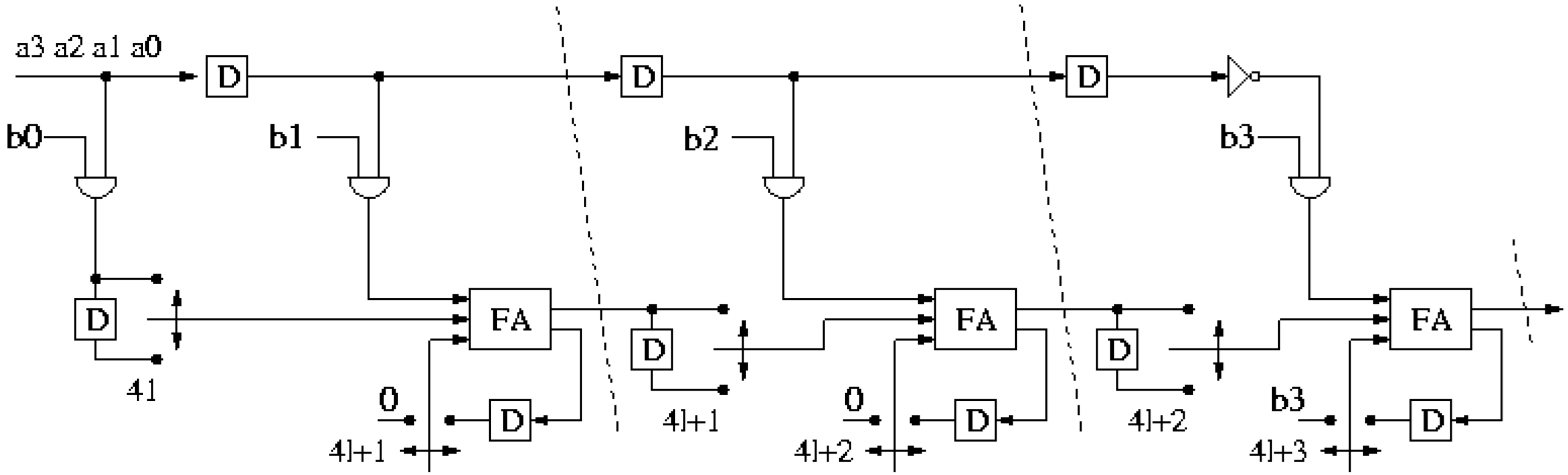
Non-causal !!





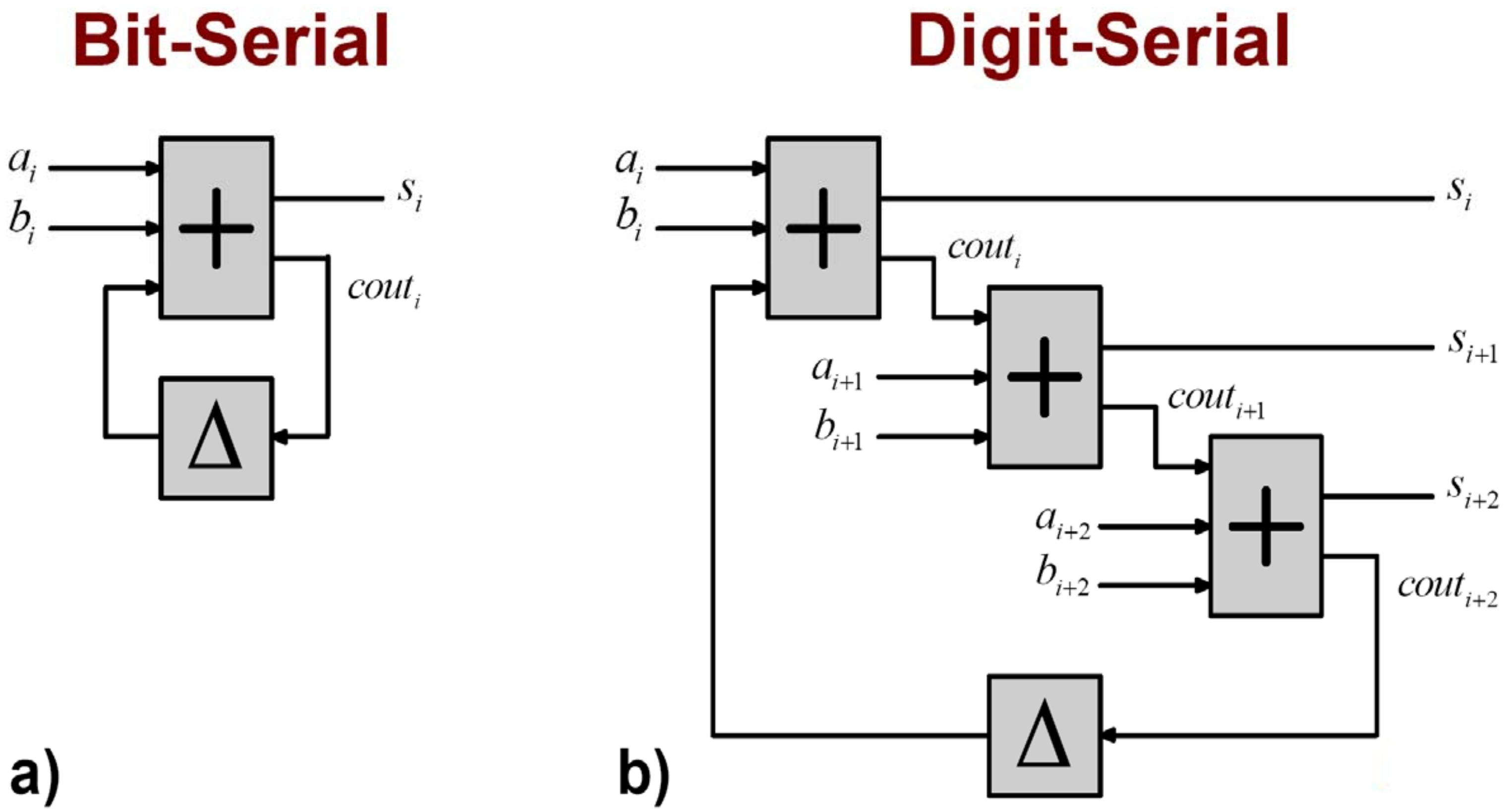
Derivation of implementable bit-serial 2's complement multiplier

The switching time instances can be derived by scheduling the bit-level computations (Fig. 13.16)

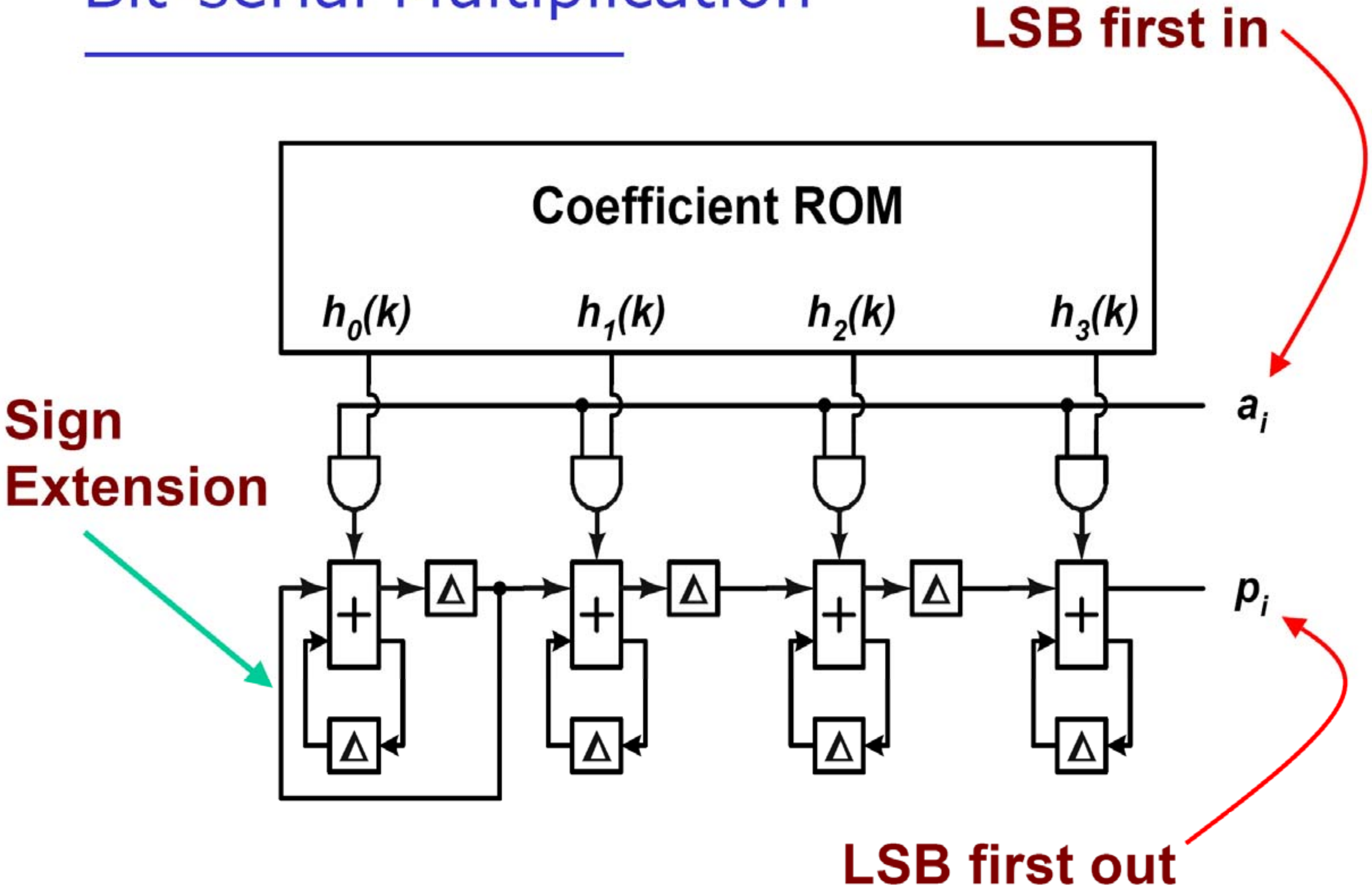


Lyon's bit-serial 2's complement multiplier

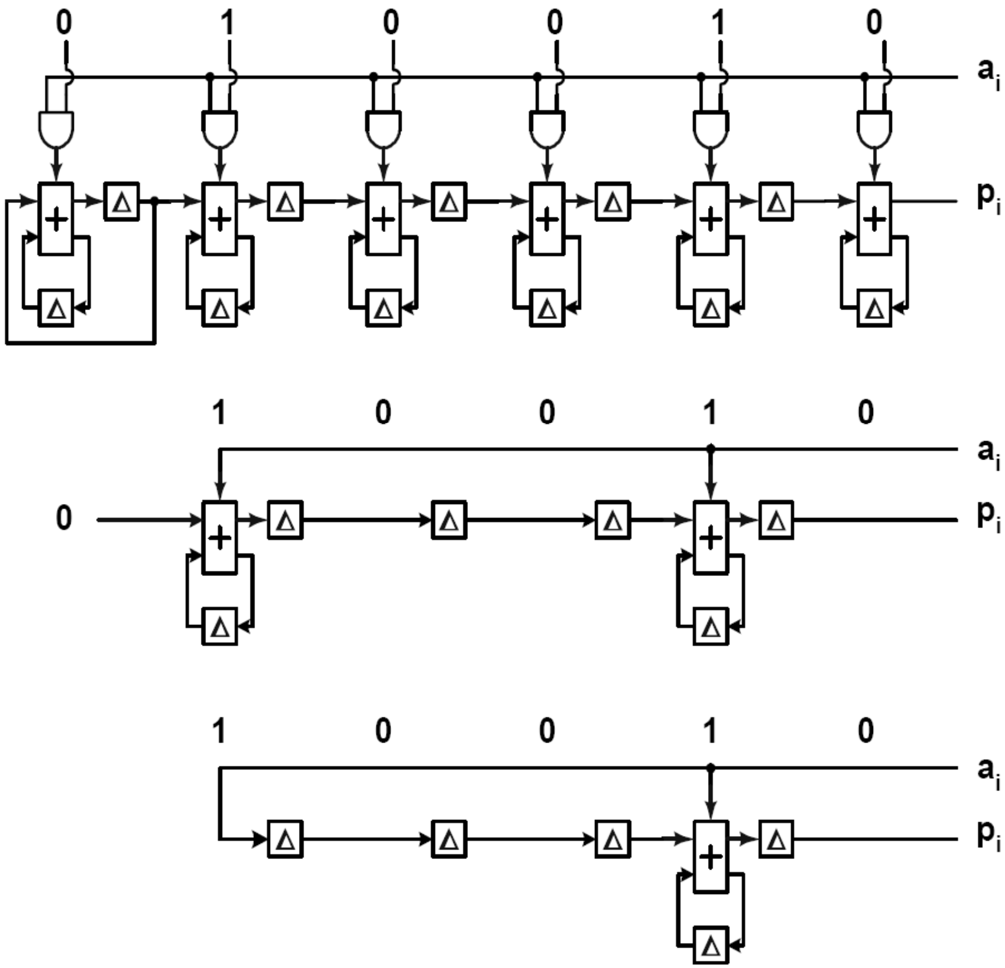
Serial Addition



Bit-serial Multiplication



Fixed Coefficient Multiplication



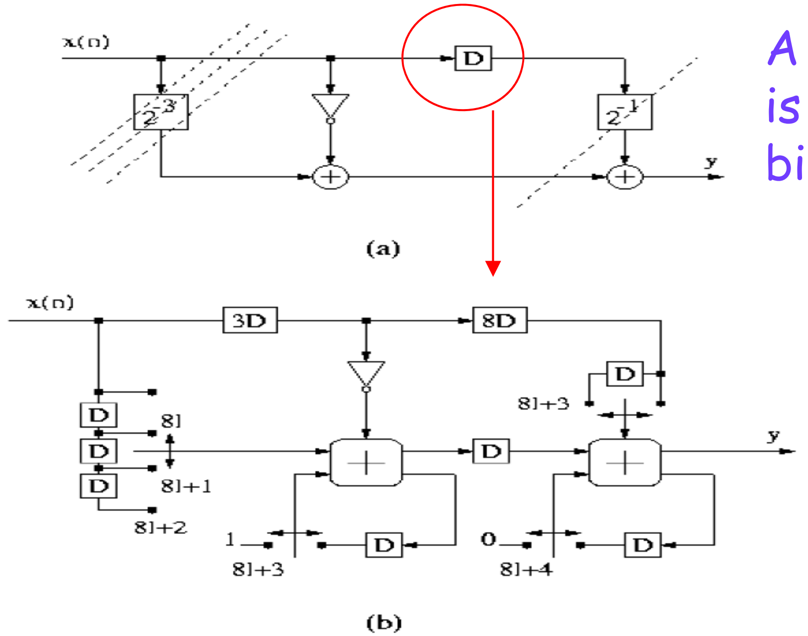
Saves more than 1/2 of the adders at an average

Bit-Serial FIR Filter

- Constant coefficients are decomposed and implemented using bit-serial shifts and adds.
- Apply feedforward cutset retiming and replacing the delayed scaling operators with switches, a feasible bit-serial pipelined bit-serial FIR filter can be derived.

Example

Wordlength=8



A word-level delay is equivalent to W bit-level delays

Bit-level pipelined bit-serial FIR filter, $y(n) = (-7/8)x(n) + (1/2)x(n-1)$, where constant coefficient multiplications are implemented as shifts and adds as $y(n) = -x(n) + x(n)2^{-3} + x(n-1)2^{-1}$.
(a)Filter architecture with scaling operators;
(b) feasible bit-level pipelined architecture

Bit-Serial IIR Filter

- Steps for deriving a bit-serial IIR filter architecture:
  - A bit-level pipelined bit-serial implementation of the FIR section needs to be derived.
  - The input signal x(n) is added to the output of the bit-serial FIR section w(n).
  - The resulting signal y(n) is connected to the signal y(n-1).
  - The number of delay elements in the edge marked ?D needs to be determined.(see figure in next page)
- For, systems containing loop, the total number of delay elements in the loops should be consistent with the original SFG, in order to maintain synchronization and correct functionality.
- **Loop delay synchronization** involves matching the number of word-level loop delay elements and that in the bit-serial architecture. The number of bit-level delay elements in the bit-serial loops should be $W \times N_D$, where W is signal word-length and N_D denotes the number of delay elements in the word-level SFG.

IIR Example

- Consider implementation of the IIR filter

$$Y(n) = (-7/8)y(n-1) + (1/2)y(n-2) + x(n)$$

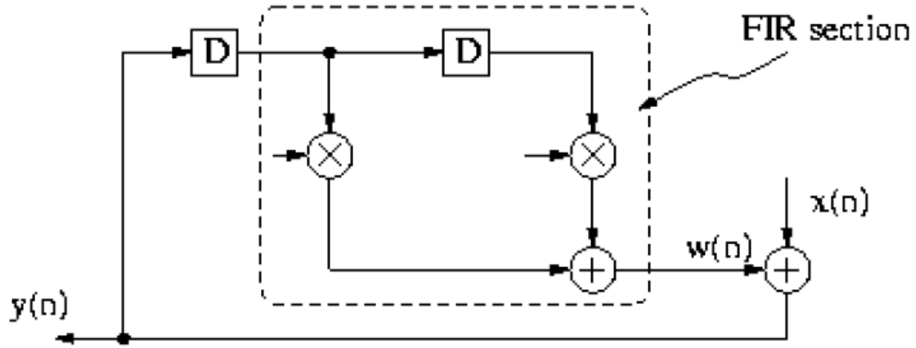
where, signal word-length is assumed to be 8.

- The filter equation can be re-written as follows:

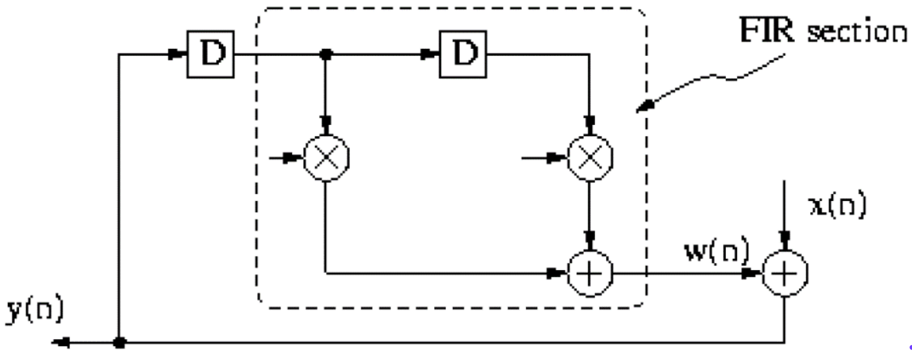
$$w(n) = (-7/8)y(n-1) + (1/2)y(n-2)$$

$$Y(n) = w(n) + x(n)$$

which can be implemented as an FIR section from y(n-1) with an addition and a feedback loop as shown below:



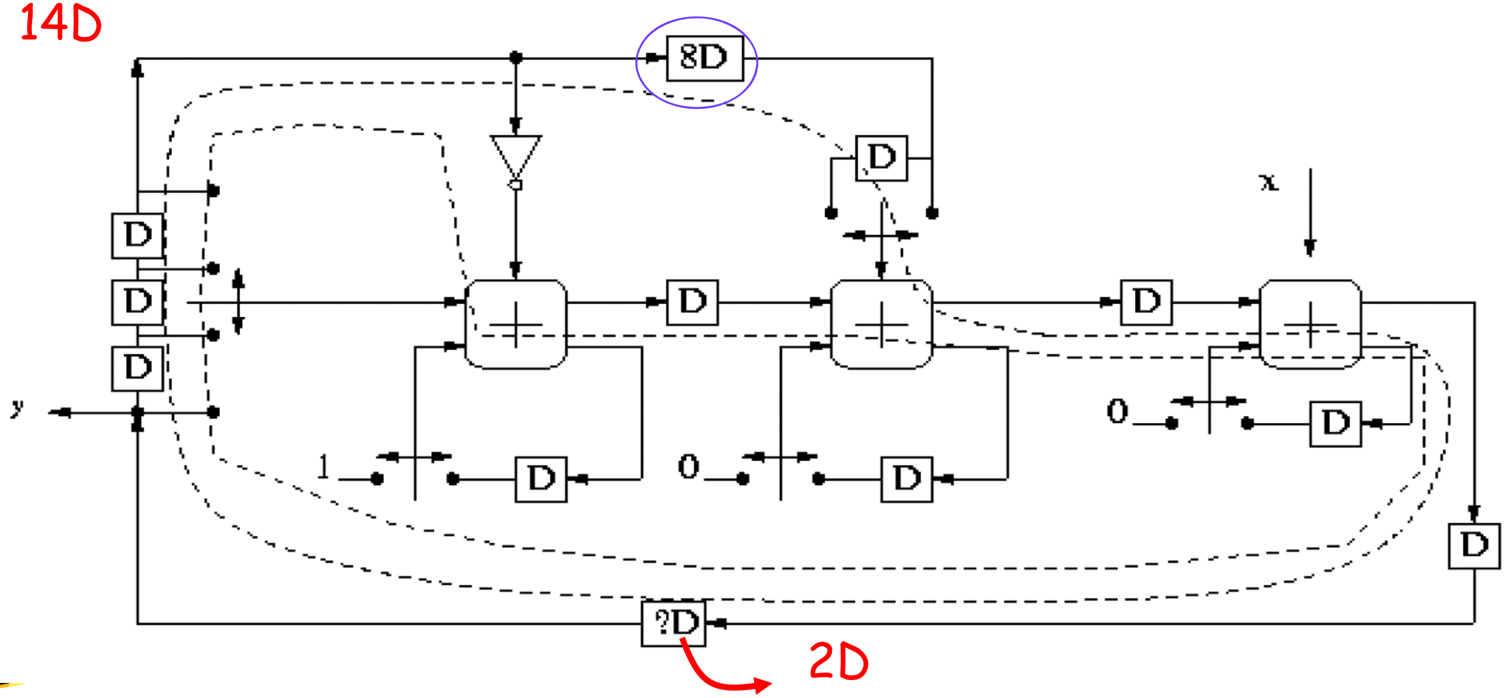
2-loops



Inner loop : 1D
Outer loop: 2D

Bit-serial implementation

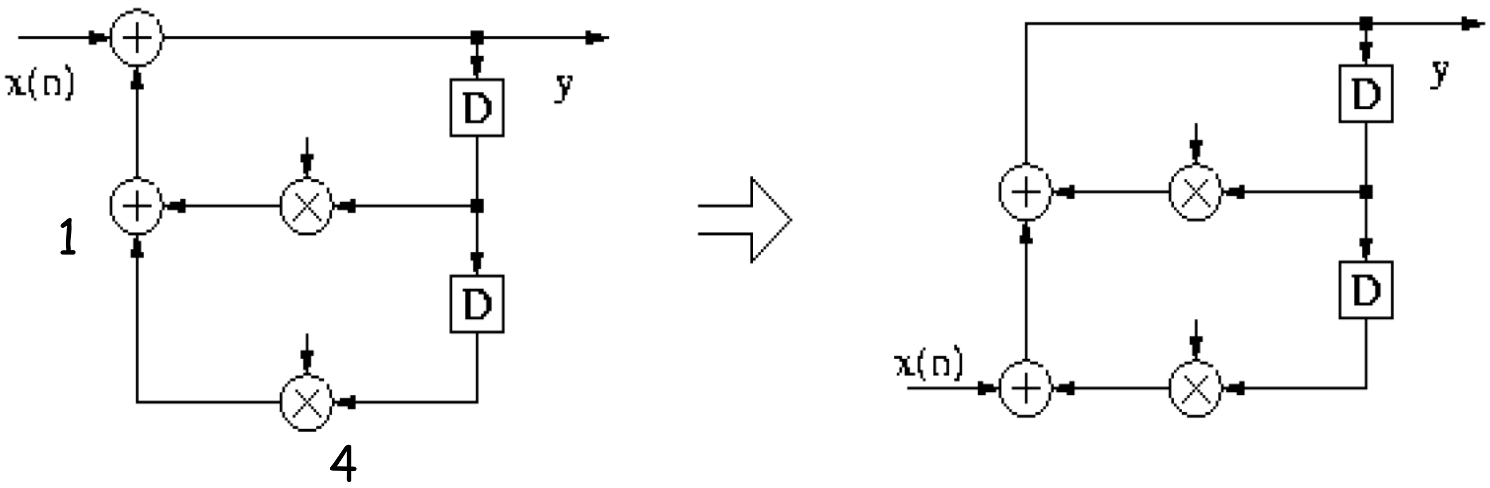
Inner : 6D
Outer: 14D



Bit-Serial IIR

- It is possible that the loops in the intermediate bit-level pipelined architecture may contain more than $W \times N_D$ number of bit-level delay elements ➔ the wordlength needs to be increased !!
- The wordlength is a parameter
- The minimum feasible wordlength of a bit-level pipelined bit-serial system decides its maximum throughput.
- The minimum feasible wordlength is constrained by the iteration bound of the SFG.
- Note: the minimum wordlength of the above example is 6

IIR Examples



T_M+2T_A

T_M+T_A

Minimum feasible wordlength=6

Throughput = 1 output/6 cycles (word)

Minimum feasible wordlength=5

Throughput = 1 output/5 cycles

Canonic Signed Digit (CSD)

- For fixed constant coefficient multiplications
- Encoding a binary number such that it contains the fewest number of non-zero bit
- Example: A sequence of ones can be replaced with
  - A "-1" at the least significant position of the sequence
  - A "1" at the position to the left of the most significant position of the sequence
  - Zeros between the "1" and the "-1"

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

It can save more than 2/3 of the adder cells in an average !!

Properties of CSD Numbers

- No 2 consecutive bits in a CSD number are non-zero.
- The CSD representation of a number contains the minimum possible number of non-zero bits, thus the name canonic.
- The CSD representation of a number is unique.
- CSD numbers cover the range (-4/3,4/3), out of which the values in the range [-1,1) are of greatest interest.
- Among the W-bit CSD numbers in the range [-1,1), the average number of non-zero bits is $W/3 + 1/9 + O(2^{-W})$. Hence, on average, CSD numbers contains about 33% fewer non-zero bits than two's complement numbers.

Please refer to the reference.

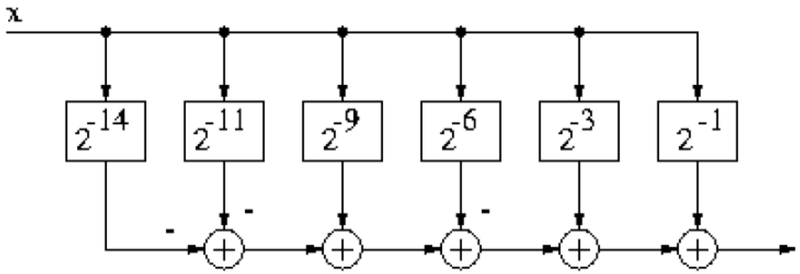
Remarks

- Booth's Modified Algorithm
  - For variable coefficients
- Canonical Signed Digit
  - For fixed coefficients
  - Optimal

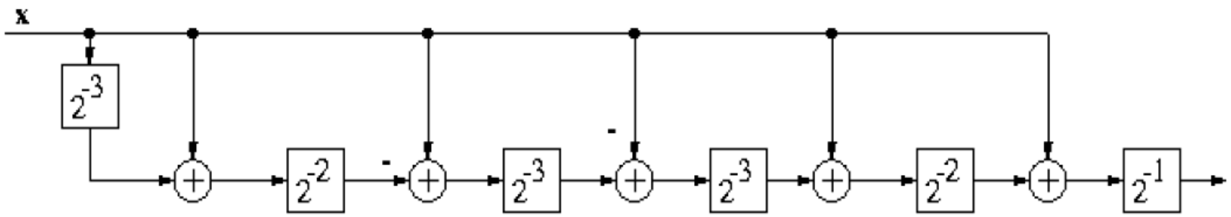
Horner's Rule for Precision Improvement

- The truncation errors introduced in the constant wordlength multiplier not only depend on the multiplicand and multiplier value, but also on the arrangement of the partial product accumulation.
- Horner's Rule: to delay the scaling operations common to the 2 partial products. That is, by applying partial products accumulation to reduce the truncation error.
- Example: $x2^{-5} + x2^{-3}$ can be implemented as $(x2^{-2}+x)2^{-3}$ to increase the accuracy

Horner's Rule Example



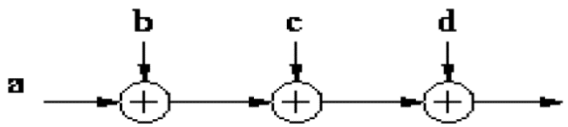
A CSD multiplier using linear arrangement of adders to compute x × 0.10100100101001



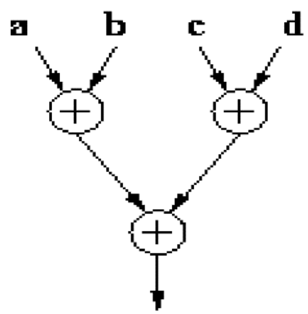
Rearrangement of the CSD multiplication of x × 0.10100100101001 using Horner's rule for partial product accumulation to reduce the truncation error.

Latency Reduction

- With the same hardware complexity, the tree type arrangement can reduce the latency from $(N-1)T_A$ to $\lceil \log_2 N \rceil T_A$



Linear arrangement



Binary tree arrangement

Distributed Arithmetic

- Usually used in summation of inner products
- E.g.

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

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

CORDIC Algorithm

- Iterative algorithm for circular rotations
  - Example: sin, cos, to derive polar coordinates, …
- No multiplication
- CORDIC
  - COordinate Rotation DIgital Computer
  - Presented by Jack E. Volder 1959

Twiddle Factor Multiplication

$$W_N^{nk} = e^{-j(2\pi nk/N)}$$

$$\triangleq e^{-j\varphi} = \cos\varphi + j\sin\varphi,$$

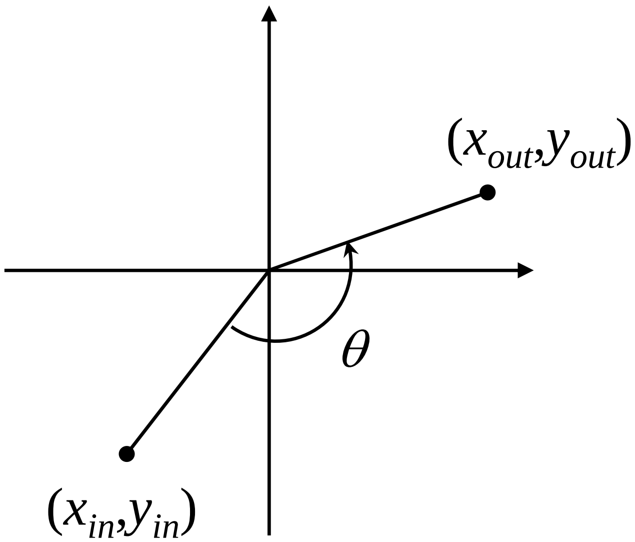
$$G = S \cdot W_N^{nk} = (s_r + j \cdot s_i) \cdot (\cos\varphi + j\sin\varphi)$$

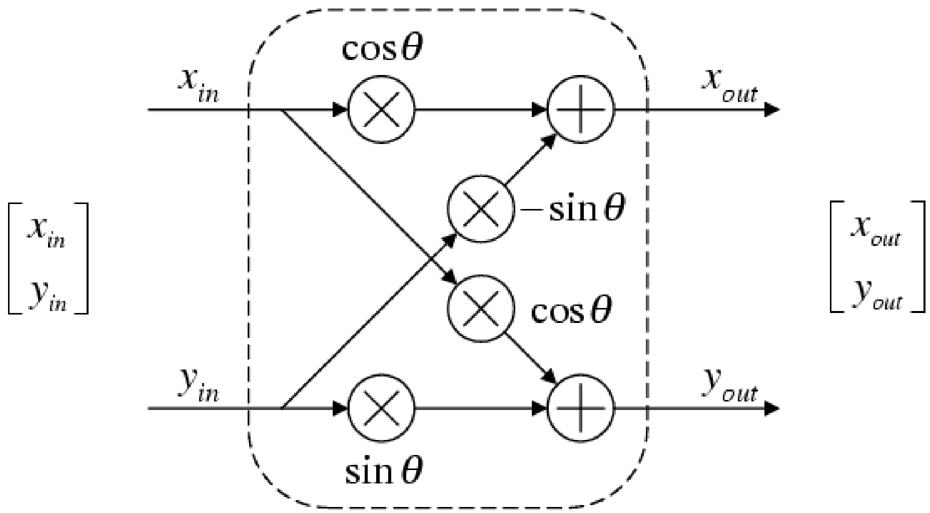
$$= (s_r \cdot \cos\varphi - s_i \cdot \sin\varphi) + j\,(s_r \cdot \sin\varphi + s_i \cdot \cos\varphi)\,.$$

$$\begin{bmatrix} g_r \\ g_i \end{bmatrix} = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix} \cdot \begin{bmatrix} s_r \\ s_i \end{bmatrix}.$$

Rotation

- Definition





Rotational Circuit

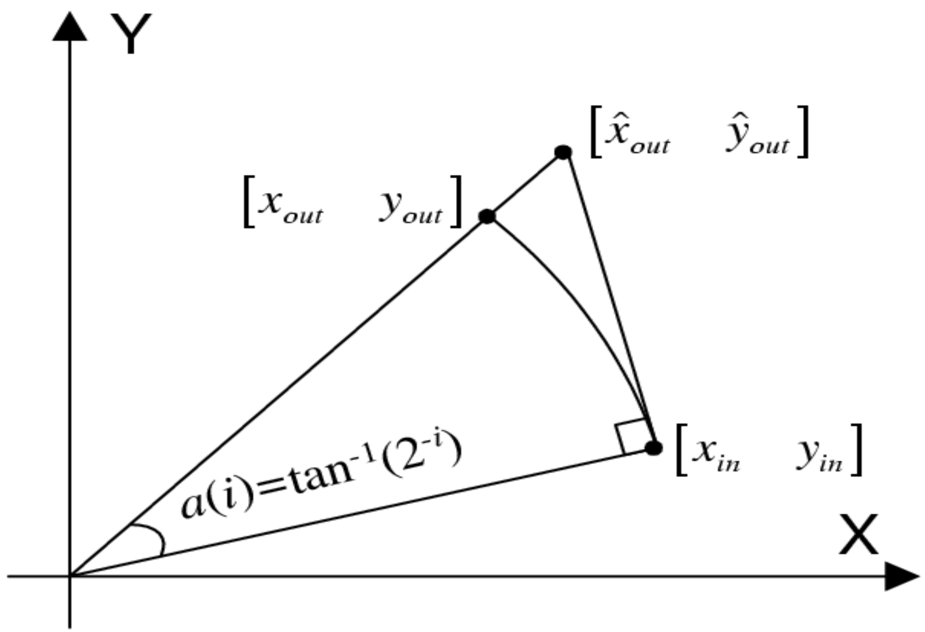
4 multiplications and 2 additions !!

$$\begin{bmatrix} x_{out} \\ y_{out} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix} \triangleq \mathbf{R} \cdot \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix}.$$

Basic Concept of CORDIC Operation

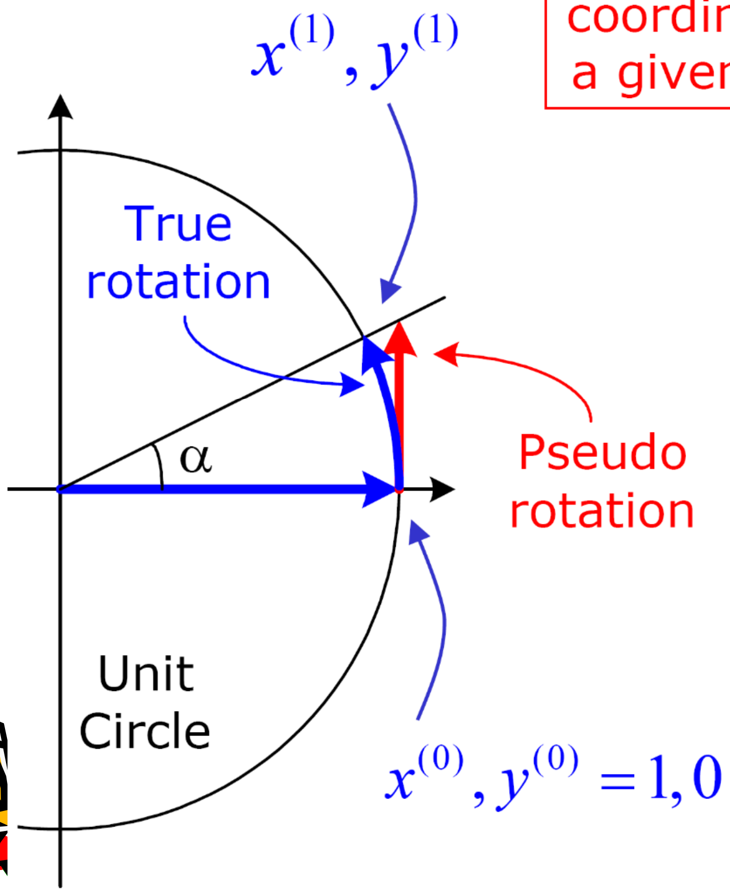
- To decompose the desired rotation angle θ into the weighted sum of a set of predefined elementary rotation angles $\theta^{(i)}$, i=0,1,2, …, W
- Consequently, the rotation through each of them can be accomplished with simple shift-and-add operation
- No Multiplication at all !!

Real Rotation v.s. Pseudo Rotation



Real Rotation

Find the x,y coordinates for a given angle



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

By definition

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

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

By definition

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

Example:

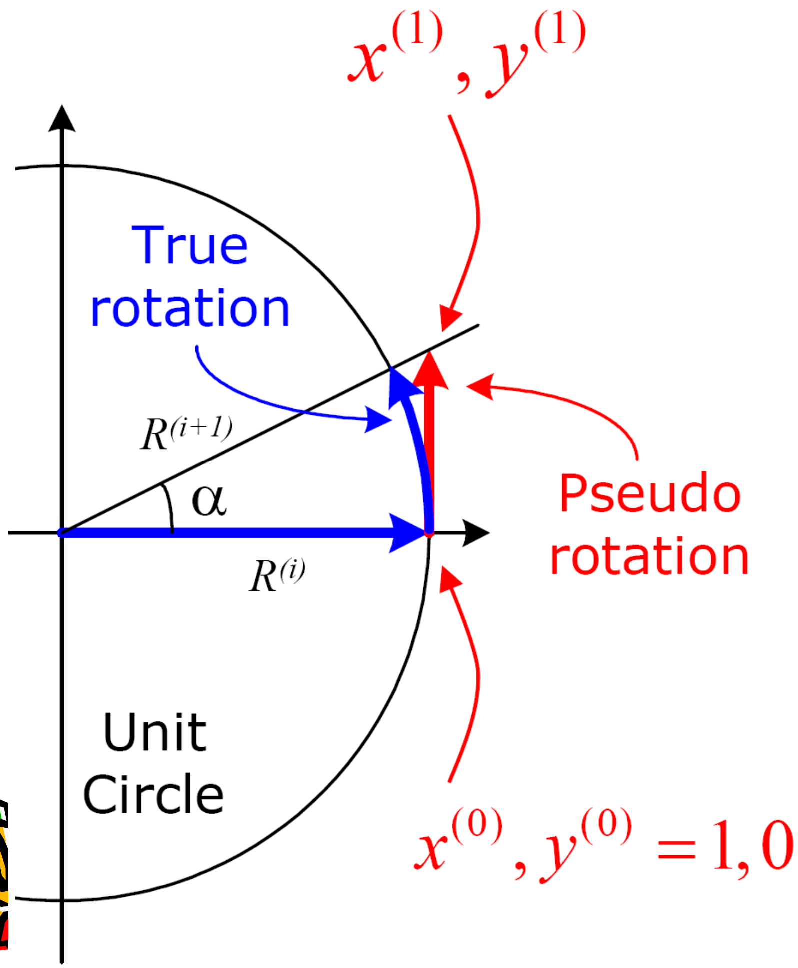
$$x^{(1)} = \frac{x^{(0)} - y^{(0)} \tan\alpha^{(i)}}{\sqrt{1+\tan^2\alpha^{(i)}}}$$

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

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

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

Pseudo Rotation



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

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

Example:

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

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

$$\boxed{\text{However the length } R > 1}$$

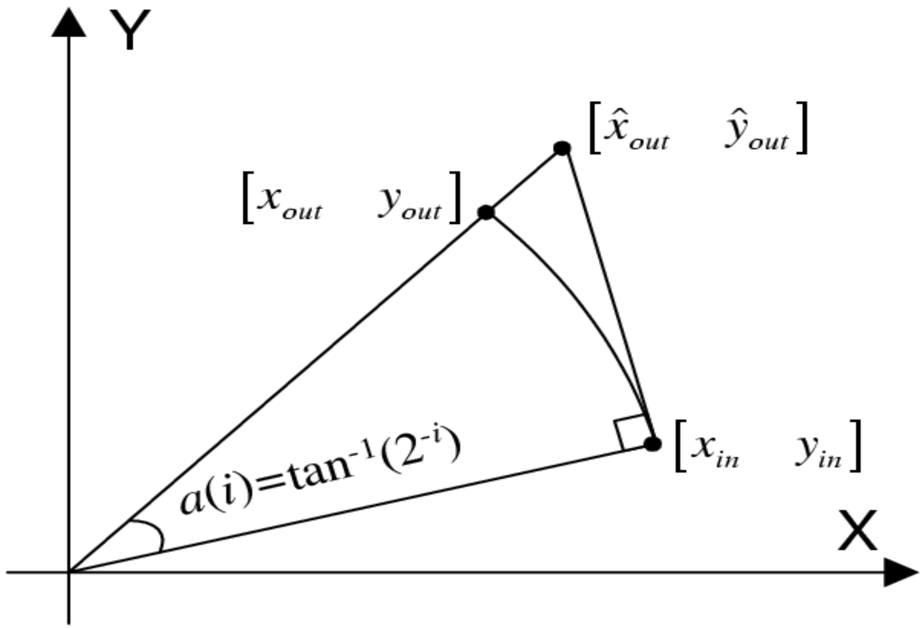
$$R^{(i+1)} = R^{(i)} \frac{1}{\cos\alpha^{(i)}} =$$

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

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

CORDIC Rotation

$$\begin{bmatrix} x_{out} \\ y_{out} \end{bmatrix} = \frac{1}{\sqrt{1+2^{-2i}}} \cdot \boxed{\begin{bmatrix} 1 & -2^{-i} \\ 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix}} \triangleq \frac{1}{\sqrt{1+2^{-2i}}} \cdot \boxed{\begin{bmatrix} \hat{x}_{out} \\ \hat{y}_{out} \end{bmatrix}}$$



Conventional CORDIC Algorithm

$$\begin{bmatrix} x^{(i+1)} \\ y^{(i+1)} \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \cdot \begin{bmatrix} x^{(i)} \\ y^{(i)} \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} x^{(i+1)} \\ y^{(i+1)} \end{bmatrix} = \cos\alpha \cdot \begin{bmatrix} 1 & -\tan\alpha \\ \tan\alpha & 1 \end{bmatrix} \cdot \begin{bmatrix} x^{(i)} \\ y^{(i)} \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} x^{(i+1)} \\ y^{(i+1)} \end{bmatrix} = \begin{bmatrix} 1 & -\mu_i 2^{-i} \\ \mu_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x^{(i)} \\ y^{(i)} \end{bmatrix}$$

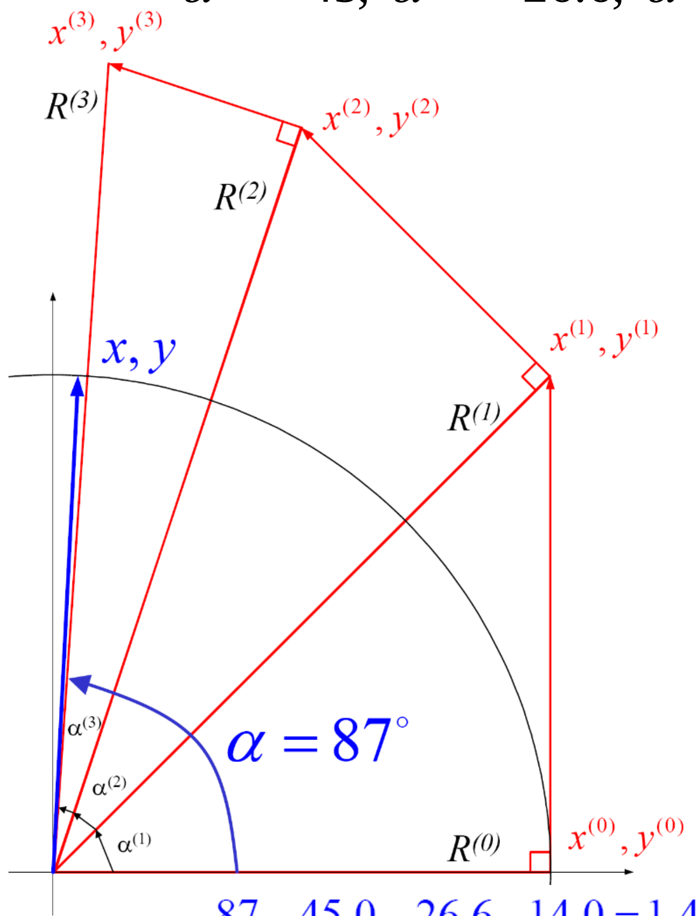
$$\textit{where} \quad \alpha_i = \tan^{-1}\left(\mu_i 2^{-i}\right) \approx \mu_i \tan^{-1}\left(2^{-i}\right)$$

Pseudo Rotation Example

- The angle α is known.
- Derive x,y using three iterations
- The vector length R is increasing during each iterations

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

$$\alpha^{(1)} = 45,\ \alpha^{(2)} = 26.6,\ \alpha^{(3)} = 14$$



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

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

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

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

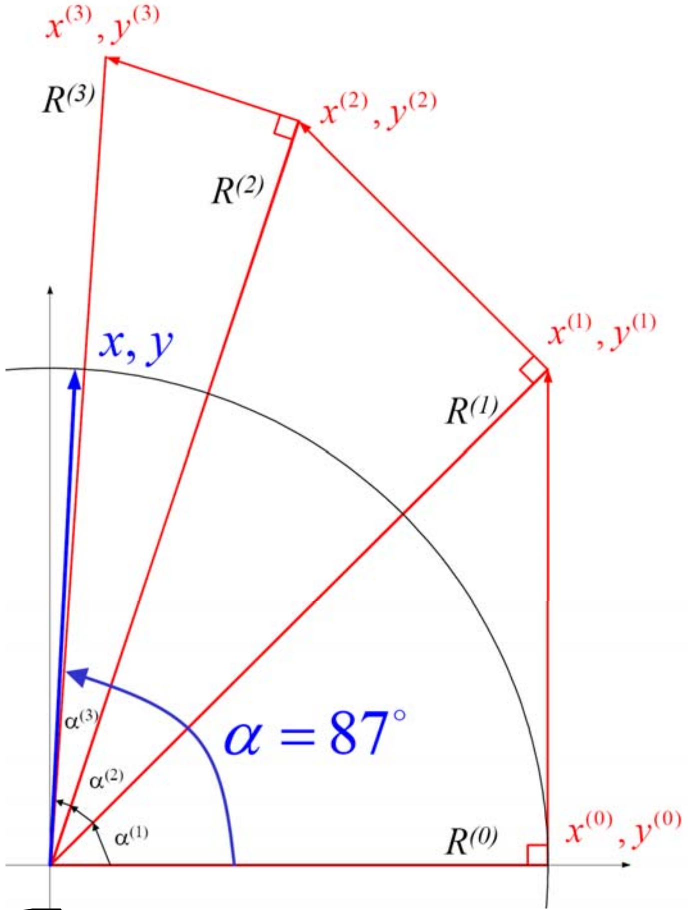
$$R^{(3)} = R^{(2)}\sqrt{1+\tan^2\alpha^{(3)}} = \sqrt{\frac{5}{2}}\sqrt{1+\tan^2 14.0^\circ} = \sqrt{\frac{85}{32}} = 1.63$$

How to Choose the Angles

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	$\tan\alpha^{(5)} = \frac{1}{16}$	$\alpha^{(0)} = \arctan\frac{1}{16} = 3.6^\circ$

CORDIC Transform

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



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

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

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

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

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

CORDIC Transform

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

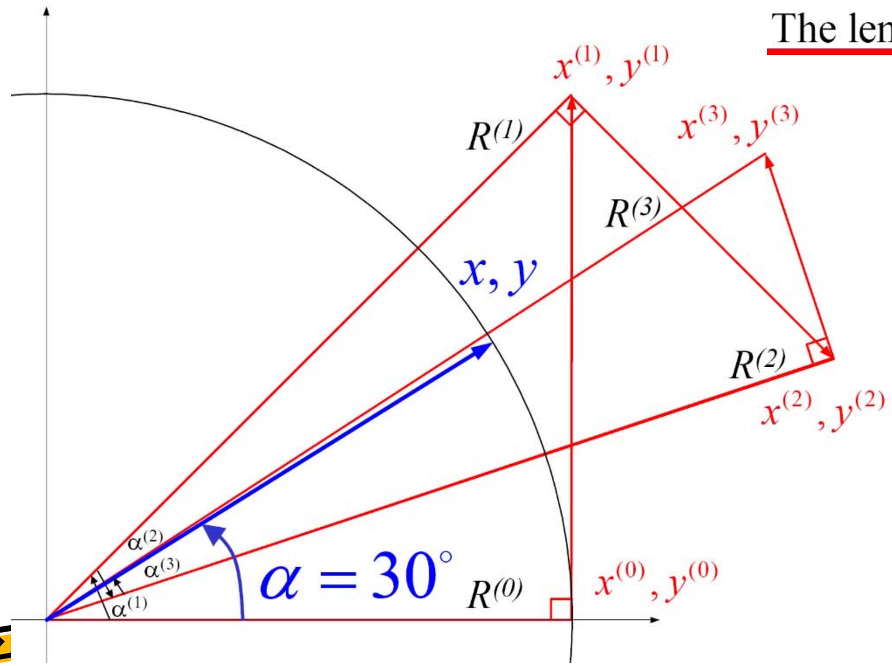
$$\alpha = 30^\circ \quad \Rightarrow \quad \text{Pos. Rot.}$$

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

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

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

The sign determines the rotation



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

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

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

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

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

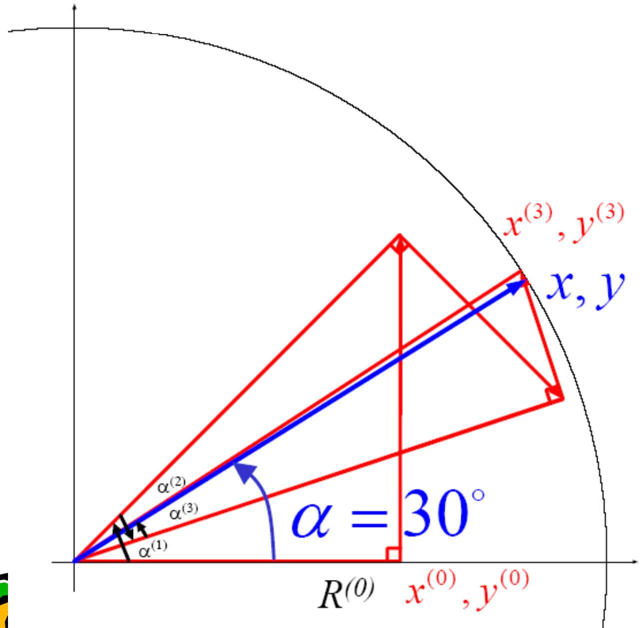
CORDIC

New start vector
(No need for multiplication)

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

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

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



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 Transform

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

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

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

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

Each CORDIC iteration require

- 3 ADD/SUB
- 2 Shifts

Elementary Angle Sets

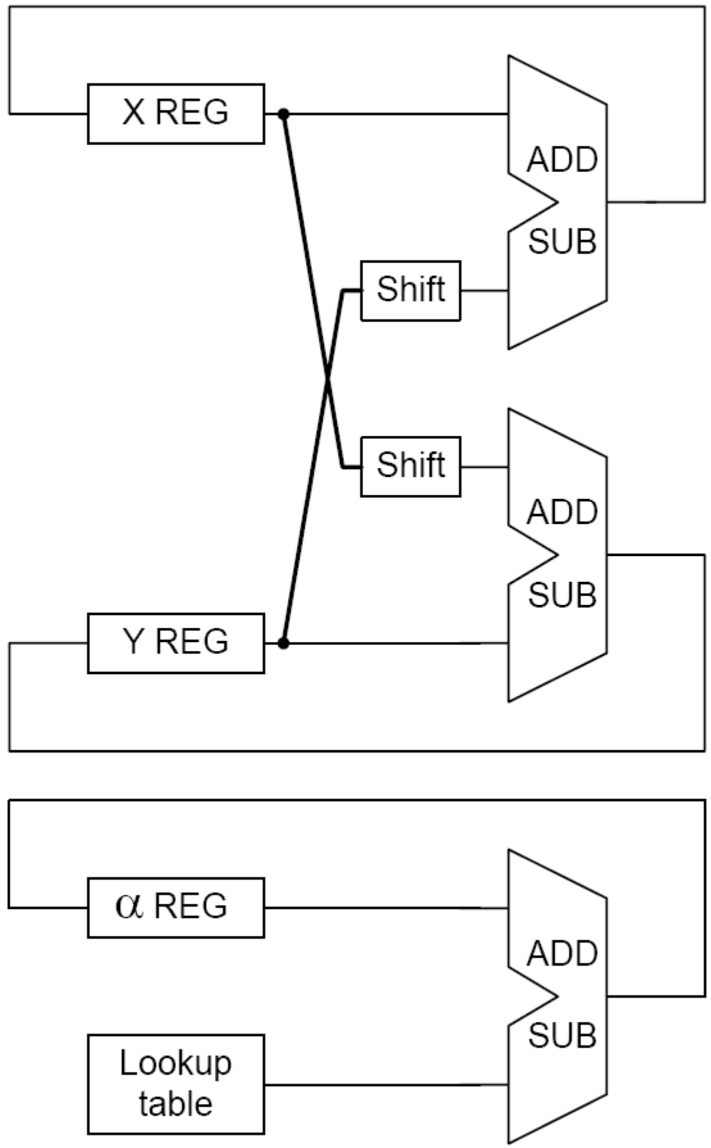
Iteration Index	Elementary Angle	Value in Radius
$i = 0$	$a(0) = \tan^{-1}(2^{-0})$	0.785398
$i = 1$	$a(1) = \tan^{-1}(2^{-1})$	0.463648
$i = 2$	$a(2) = \tan^{-1}(2^{-2})$	0.244979
$i = 3$	$a(3) = \tan^{-1}(2^{-3})$	0.124355
$i = 4$	$a(4) = \tan^{-1}(2^{-4})$	0.062419
$i = 5$	$a(5) = \tan^{-1}(2^{-5})$	0.031240
$i = 6$	$a(6) = \tan^{-1}(2^{-6})$	0.015624
$i = 7$	$a(7) = \tan^{-1}(2^{-7})$	0.007812

Conventional CORDIC with N=W=8

CORDIC Harware

Each CORDIC iteration require

- 3 ADD/SUB
- 2 Shifts



CORDIC Summary

- +++:
  - Simple shift-and-add operation
  - Small area
- ---:
  - It needs n iterations to obtain n-bit precision
  - Slow carry-propagate addition
  - Area consuming shifts (Barrel Shifter)

Summary of CORDIC Algorithm

❖ Both micro-rotation and scaling phases

% Initialization

Given $x(0), y(0)$, and $z(0)$

% Micro-rotation phase

FOR $i = 0$ to $N-1$

$$\begin{bmatrix} x(i+1) \\ y(i+1) \end{bmatrix} = \begin{bmatrix} 1 & -\mu(i)2^{-i} \\ \mu(i)2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x(i) \\ y(i) \end{bmatrix}$$

% Angle updating

$z(i+1) = z(i) - \mu(i)a(i)$, where $a(i) = \arctan(2^{-i})$

END

% Scaling phase

$$\begin{bmatrix} x_f \\ y_f \end{bmatrix} = P \begin{bmatrix} x(N) \\ y(N) \end{bmatrix} = \frac{1}{\prod_{i=0}^{N-1} \sqrt{1+2^{-2i}}} \begin{bmatrix} x(N) \\ y(N) \end{bmatrix}$$

Ref: Y. H. Hu, "CORDIC-based VLSI architecture for digital signal processing," IEEE Signal Processing Mag., pp. 16-35, July 1992.

Modes of CORDIC Operations

- Vector rotation mode (θ is given)
  - The objective is to compute the final vector $[x_f, y_f]^T$

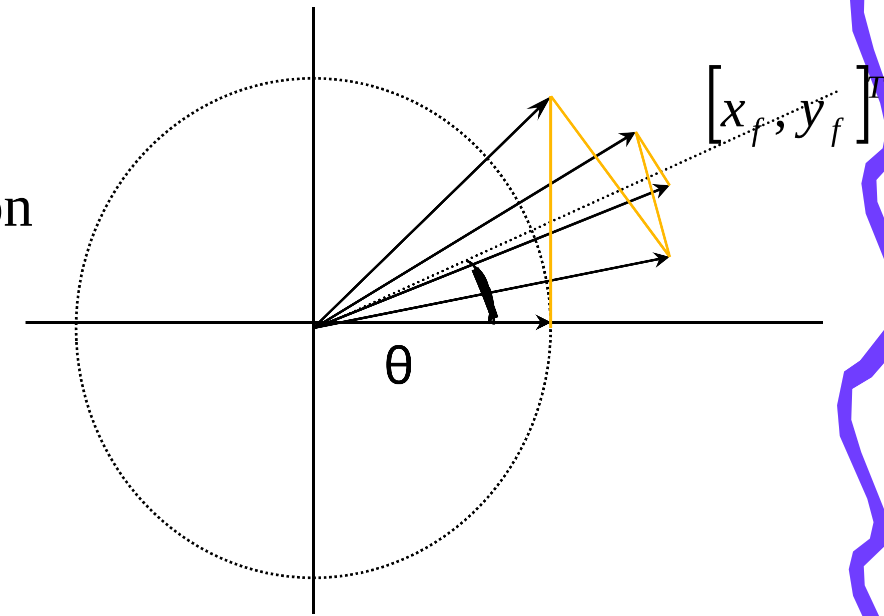
- Usually , we set z(0)= θ

$$z(0)\text{-}z(N)=\theta\text{-}z(N)=\sum_{i=0}^{N-1}\mu_i a_m(i)$$

$$|\theta - z(N)| \to 0 \text{ after the } N\text{-th iteration}$$

Rotational Sequence

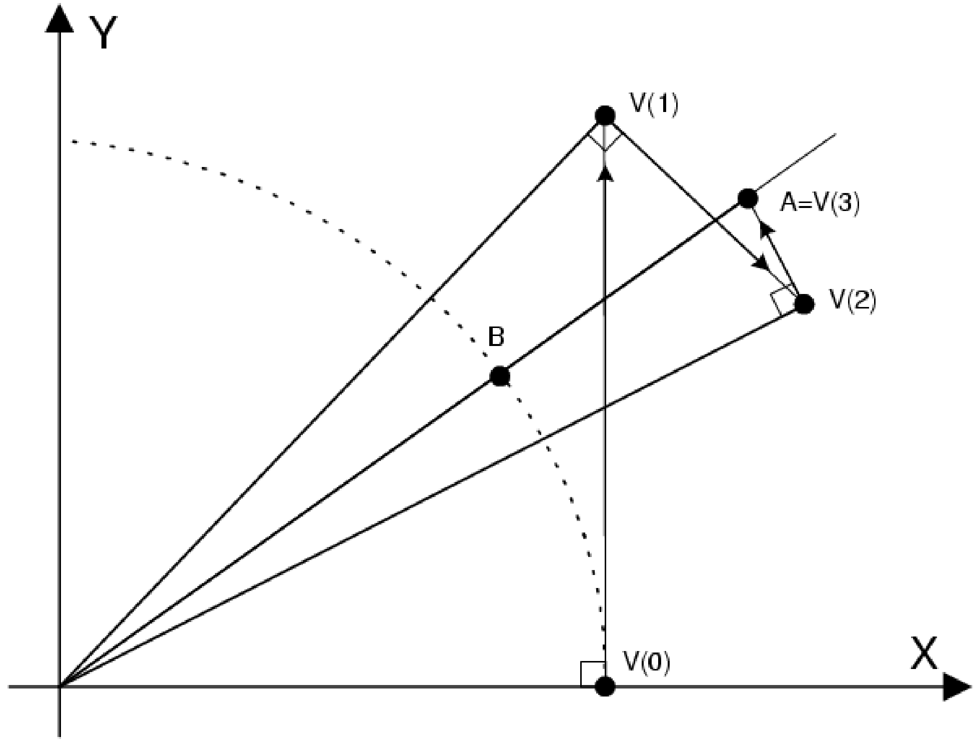
μ_i= sign of z(i)



Scaling Operation

- Scaling operation is used to maintain the "norm" of the original vector

$$P = \left(\prod_{i=0}^{R_m - 1} \sqrt{1 + 2^{-2s(i)}} \right)^{-1}.$$

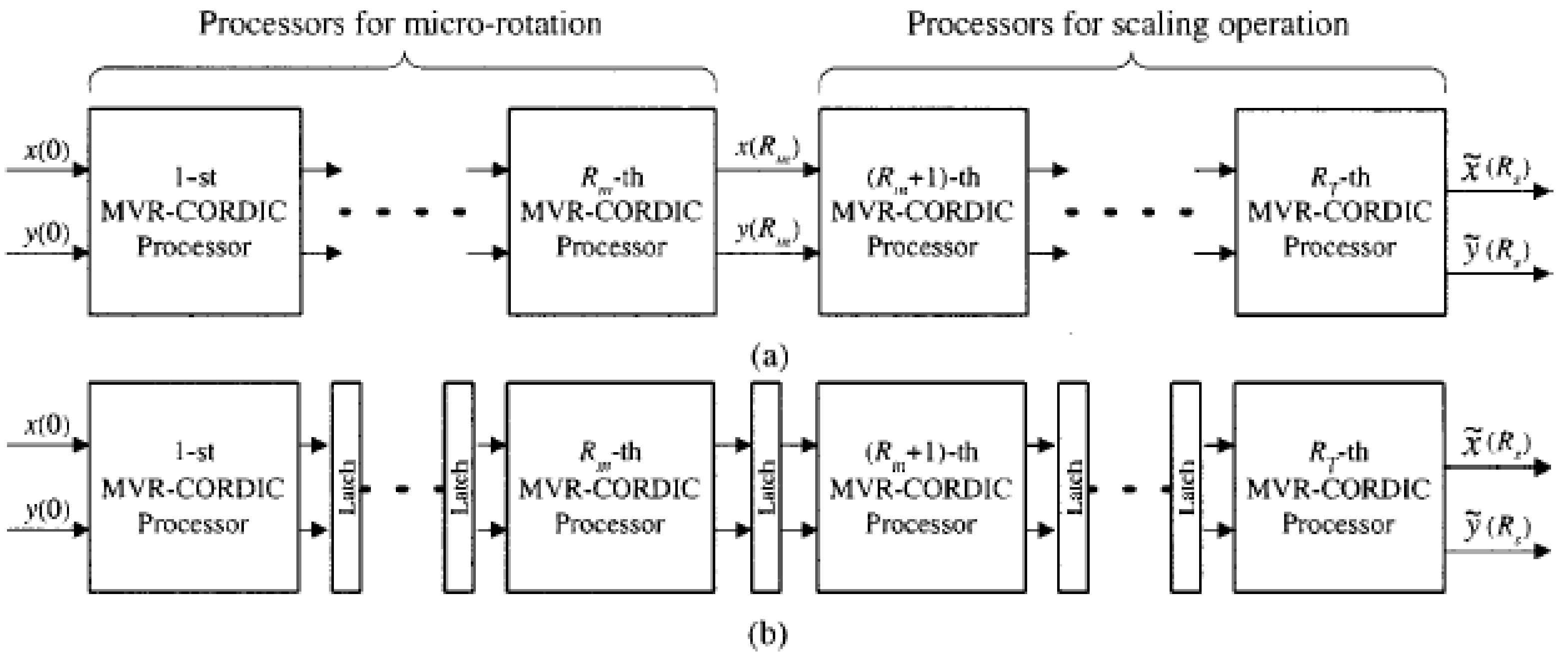


Enhancement of CORDIC

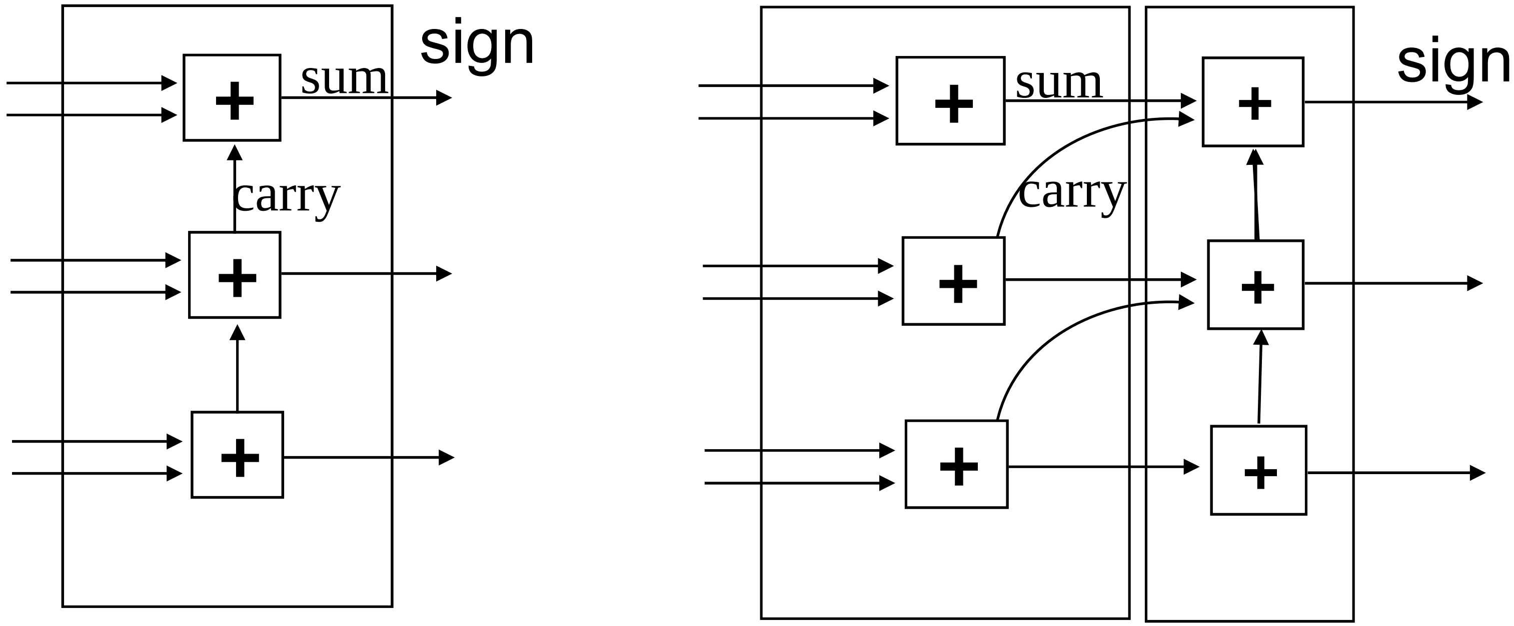
- Architecture
  - Pipelined architecture
  - Faster adder
- Algorithm
  - Radix-4 CORDIC
  - MVR-CORDIC
  - EEAS-CORDIC

Pipelined architecture

- Expand folded CORDIC processor to achieve the pipelined architecture
- Shifting can be realized by wiring



Faster Adder (CSA)

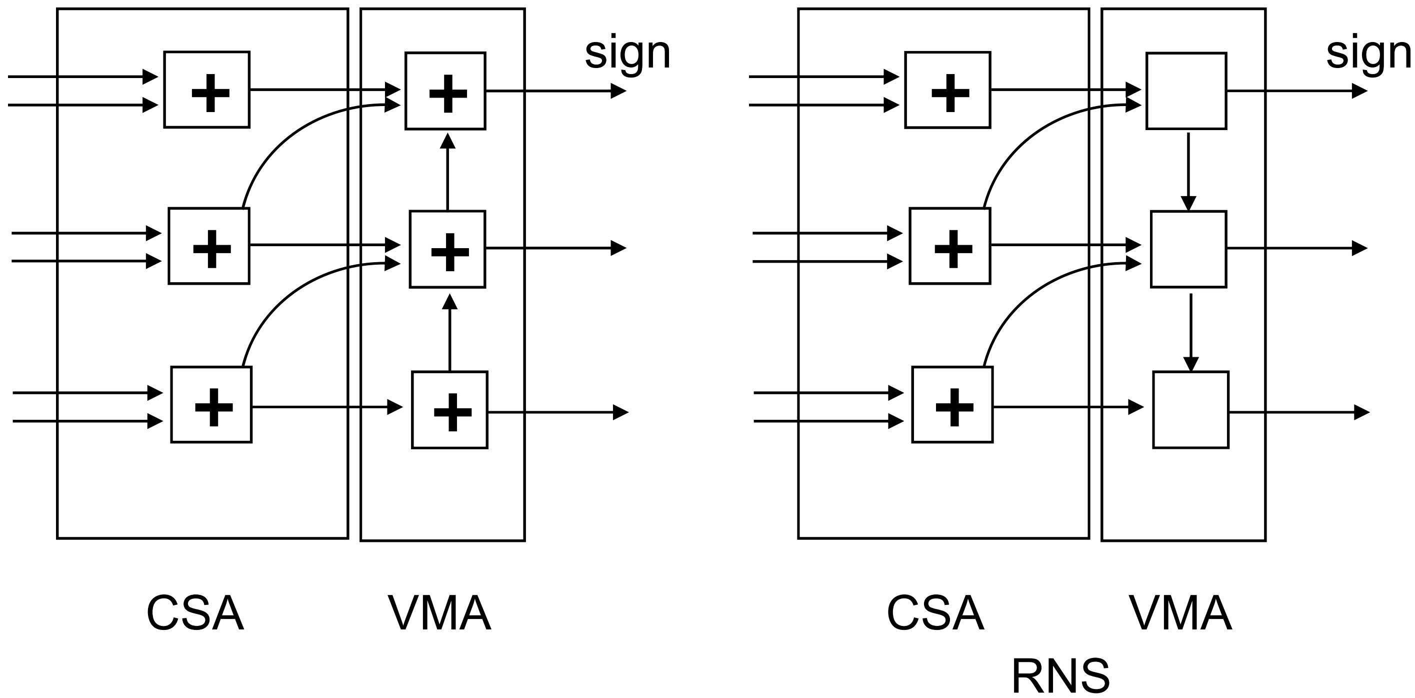


Ripple Adder
and its sign calculation

CSA VMA
and its sign calculation

Critical Path of CSA

- In on-line approach, we want to get sign bit as soon as possible !
- *Redundant Number System* to be used to obtain the sign bit quickly



Radix-4 CORDIC

- Reduce Iteration Numbers
  - High radix CORDIC.(e.g. Radix-4, Radix-8)
- 1 stage of Radix-4 = two stages of Radix-2
  - Faster computation at higher cost
- Employ the Radix-4 micro-rotations to
  - Reduce the stage number.

Modified Vector Rotation (MVR)

- Skip some micro-rotation angles
  - For certain angles, we can only reduce the iteration number but also improve the error performance.
  - For example, $\theta=\pi/4$

Conventional CORDIC $\bar{\mu}=[1, \quad -1, \quad \underline{1, \quad 1, \quad 1,} \quad \cdots]$

$\Rightarrow$ $\xi_m = 7.2*10^{-3}$

MVR-CORDIC $\bar{\mu}=[1, \quad 0, \quad 0, \quad 0, \quad 0, \quad \cdots]$

$\Rightarrow$ $\xi_m = 0$

MVR CORDIC Algorithm

- Repeat some micro-rotation angles
  - Each micro-rotation angle can be performed repeatedly
  - For example, $\theta = \pi/2$: execute the micro-rotation of a(0) twice
- Confine the number of micro-rotations to R_m
  - In conventional CORDIC, number of iteration=W
  - In the MVR-CORDIC, $R_m \ll W$
  - Hardware/timing alignment

MVR CORDIC Algorithm

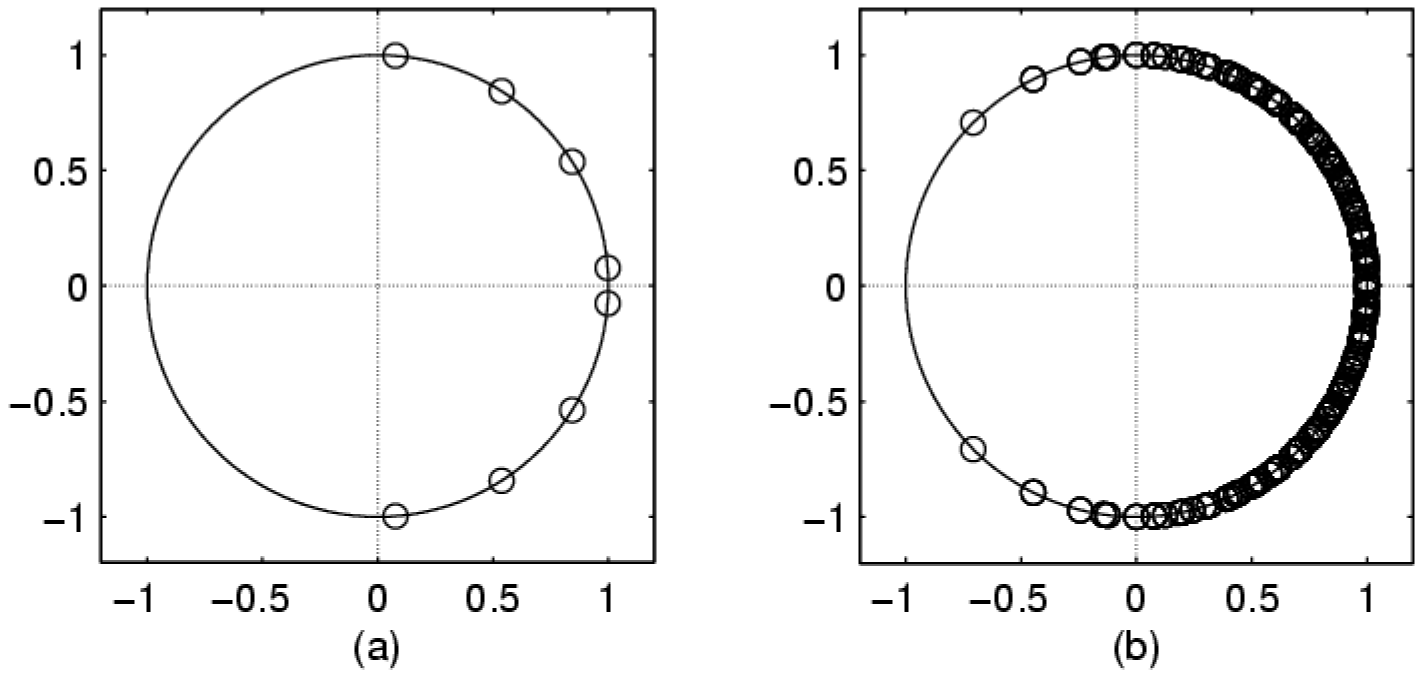
- With above three modification

$$\theta = \sum_{i=0}^{R_m - 1} \alpha(i)\theta\left(s(i)\right) + \xi_m,$$

where

- $s(i) \in \{0, 1, 2, \ldots, W\}$ is the rotational sequence that determines the micro-rotation angle in the i^{th} iteration
- $\alpha(i) \in \{-1, 0, 1\}$ is the directional sequence that controls the direction of the i^{th} micro-rotation

Constellation of Reachable Angles



(a) Conventional CORDIC with N=W=4
(b) MVR-CORDIC with W=4 and R_m=3

Extended Elementary Angle Set (EEAS)

- Apply relaxation on elementary angle set (EAS) of

$$\mathcal{S}_1 = \left\{ \tan^{-1}\left(\alpha^\star \cdot 2^{-s^\star}\right) : \alpha^\star \in \{-1, 0, 1\},\ s^\star \in \{0, 1, \cdots, N-1\} \right\}.$$

- EAS is comprised of arctangent of single singed-power-of-two (SPT) term
- Effective way to extend the EAS is to employ more SPT terms

$$\mathcal{S}_2 = \left\{ \tan^{-1}\left(\alpha_0^\star \cdot 2^{-s_0^\star} + \alpha_1^\star \cdot 2^{-s_1^\star}\right) : \right.$$

$$\left. \alpha_0^\star, \alpha_1^\star \in \{-1, 0, 1\},\ s_0^\star, s_1^\star \in \{0, 1, \cdots, W-1\} \right\}.$$

Example of EAS and EEAS

Elementary Angle	Value in Radius	Elementary Angle	Value in Radius
$r(1)=\tan^{-1}(-2^{-0})$	-0.785398	$r(5)=\tan^{-1}(2^{-2})$	0.244979
$r(2)=\tan^{-1}(-2^{-1})$	-0.463648	$r(6)=\tan^{-1}(2^{-1})$	0.463648
$r(3)=\tan^{-1}(-2^{-2})$	-0.244977	$r(7)=\tan^{-1}(2^{-0})$	0.785398
$r(4)=\tan^{-1}(0)$	0.0		

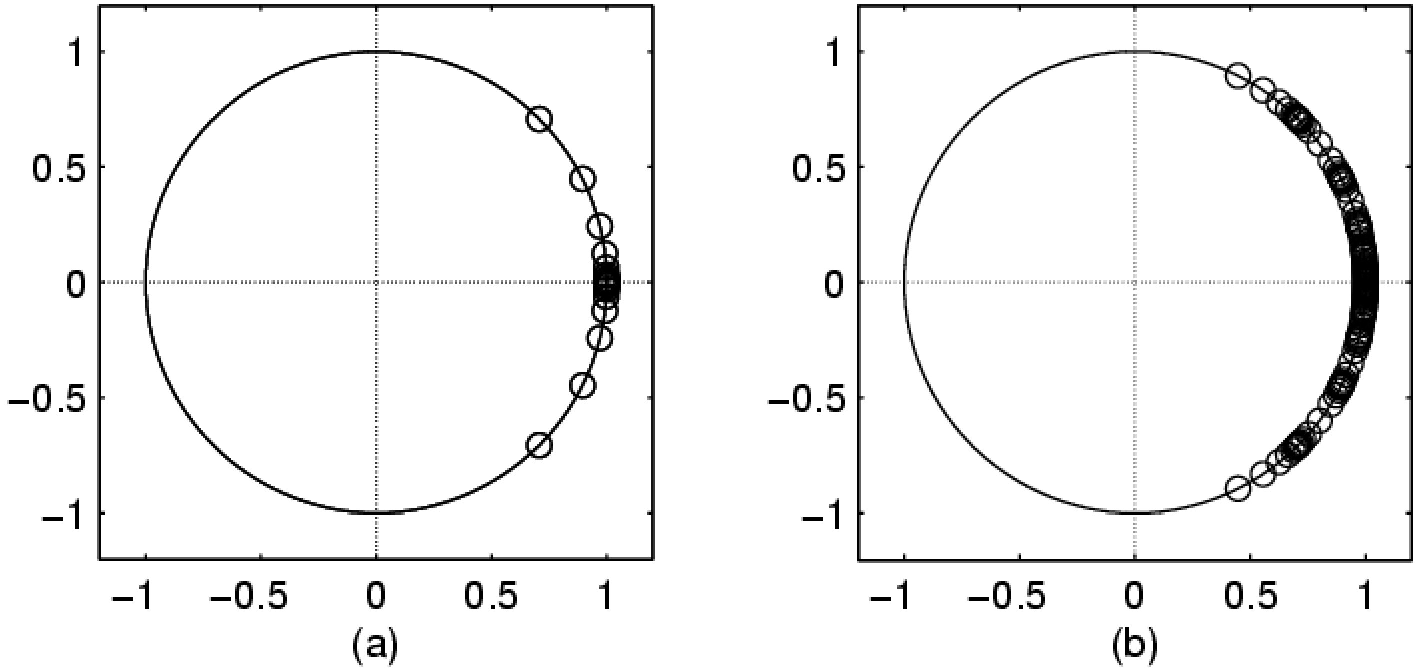
(a)

Elementary Angle	Value in Radius	Elementary Angle	Value in Radius
$r(1)=\tan^{-1}(-2^{-0}-2^{-0})$	-1.107149	$r(9)=\tan^{-1}(2^{-2})$	0.244979
$r(2)=\tan^{-1}(-2^{-0}-2^{-1})$	-0.982793	$r(10)=\tan^{-1}(2^{-1})$	0.463648
$r(3)=\tan^{-1}(-2^{-0}-2^{-2})$	-0.896055	$r(11)=\tan^{-1}(2^{-1}+2^{-2})$	0.643501
$r(4)=\tan^{-1}(-2^{-0})$	-0.785398	$r(12)=\tan^{-1}(2^{-0})$	0.785398
$r(5)=\tan^{-1}(-2^{-1}-2^{-2})$	-0.643501	$r(13)=\tan^{-1}(2^{-0}+2^{-2})$	0.896055
$r(6)=\tan^{-1}(-2^{-1})$	-0.463647	$r(14)=\tan^{-1}(2^{-0}+2^{-1})$	0.982793
$r(7)=\tan^{-1}(-2^{-2})$	-0.244979	$r(15)=\tan^{-1}(2^{-0}+2^{-0})$	1.107149
$r(8)=\tan^{-1}(0)$	0.0		

(b)

Example of elementary angles of (a) EAS S_1, (b) EEAS S_2, with wordlength $W=3$.

Constellation of EEAS

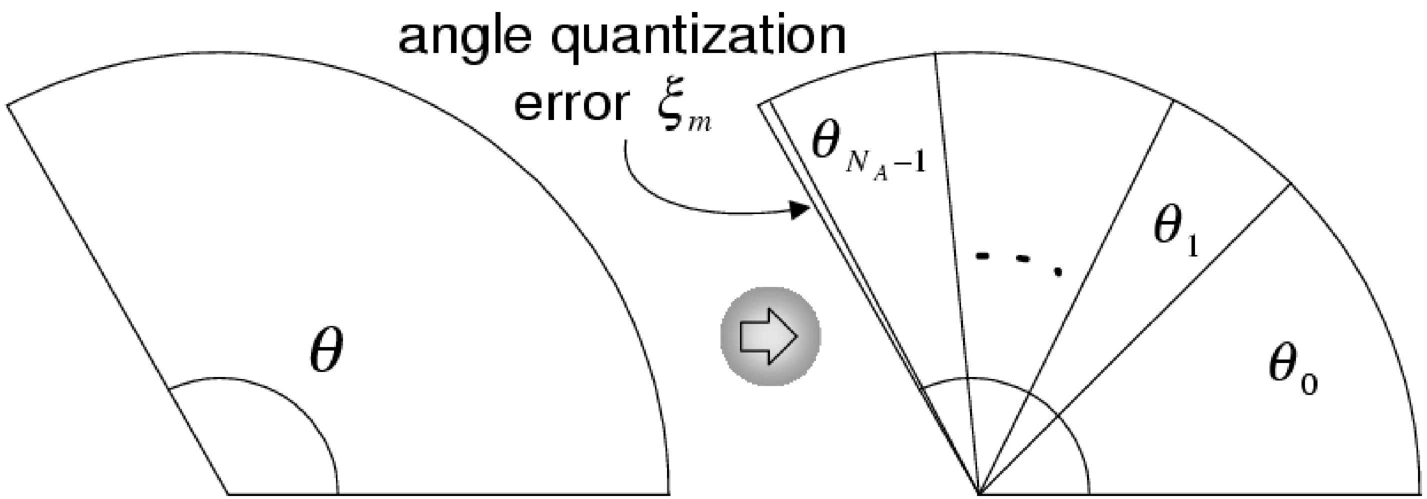


Constellation of elementary angles of (a) EAS S_1,
(b) EEAS S_2, with wordlength W=8.

Angle Quantization

- Approximation is applied on "ANGLE"
- Decompose target angle into sub-angles

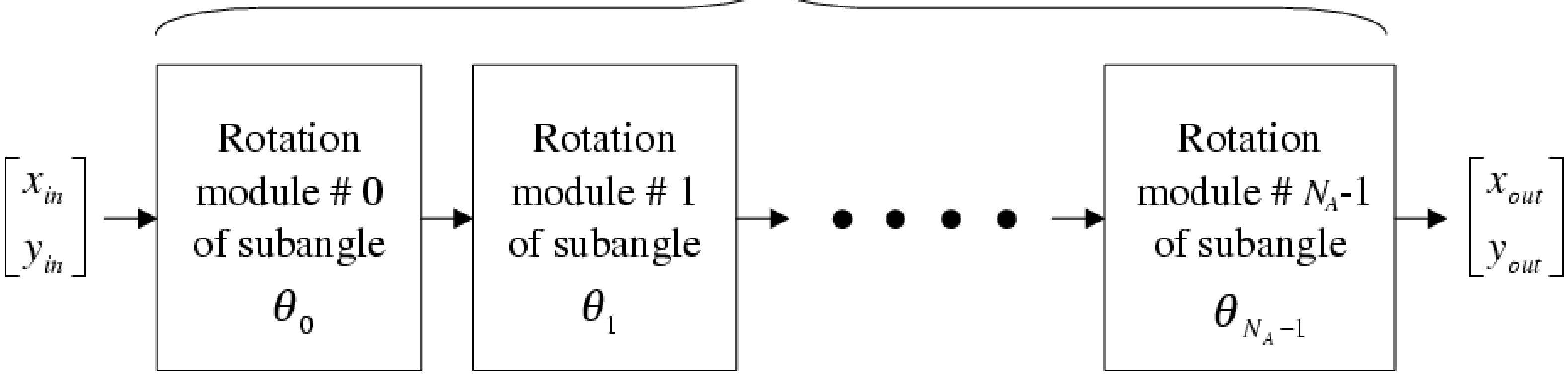
$$\xi_m \triangleq \theta - \sum_{i=0}^{N_A-1} \theta_i,$$



Angle Quantization (Cont.)

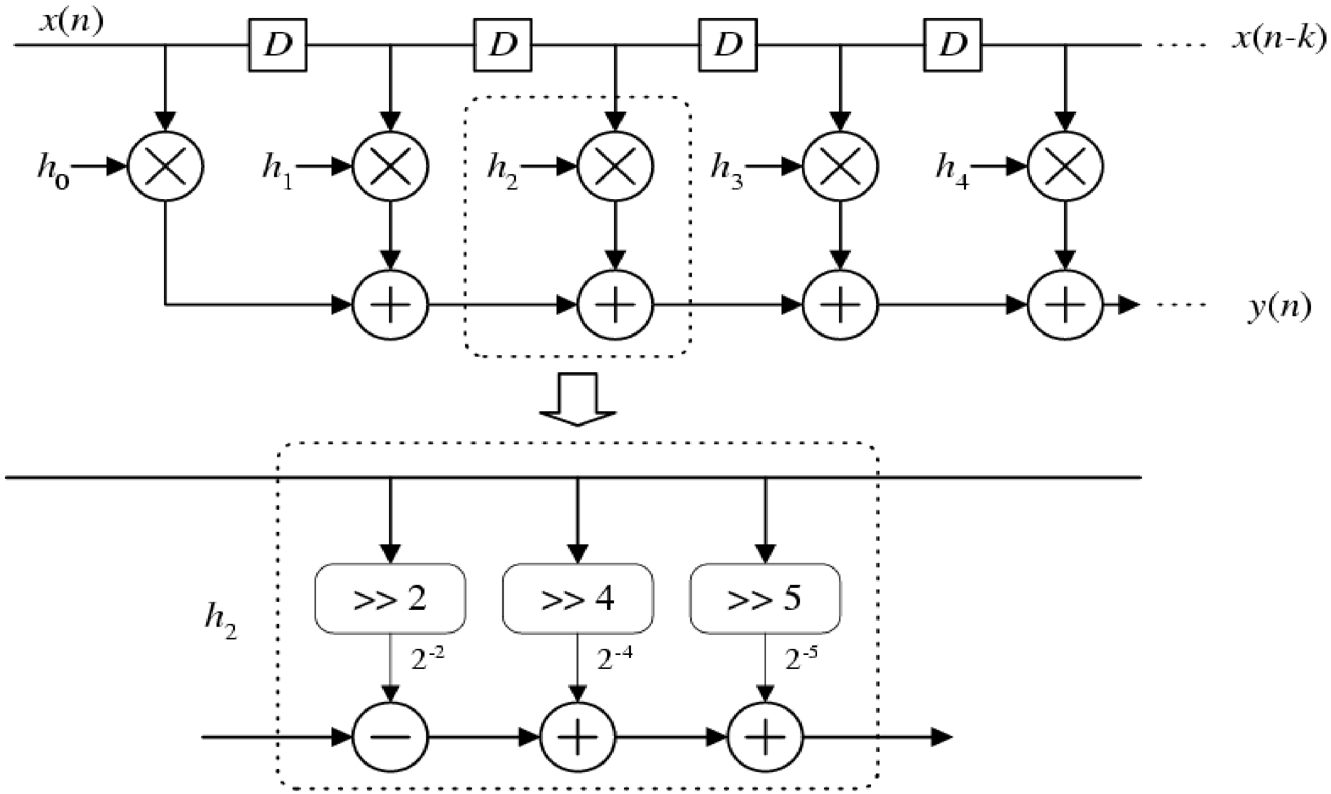
- Two key design issues in AQ process
  1) Determine (construct) the sub-angle, and each sub-angle needs to be easy-to-implementation in practical implementation.
  2) How to combine sub-angles such that angle quantization can be minimized.

$$\theta \approx \theta_0 + \theta_1 + \ldots + \theta_{N_A - 1}$$



Angle Quantization (Cont.)

- AQ process is conceptually similar to Sum-of-Power-of-Two (SPT) quantization



Angle Quantization (Cont.)

- Correspondences between AQ process and SPT (CSD) quantization process

	Canonical Signed Digit (CSD)	Angle Quantization (AQ)
Target of Approximation	Coefficient, h_k	Rotation angle, θ
Basic element	Non-zero digit, 2^{-i}	Sub-angle, θ_i
Basic operation	Shift-and-add operation	Shift-and-add operation
Approximation equation	$h_k \approx \sum_{j=0}^{N_D-1} g_j \cdot 2^{-d_j}$	$\theta \approx \sum_{j=0}^{N_A-1} \theta_j$

Conclusion

- Rotational Engine plays an important role in modern DSP systems
- The conventional CORDIC is an interesting idea to reduce VLSI hardware cost in implementing the rotation operation (as well as other arithmetic operations).
- The idea of Angle Quantization is firstly introduced in Ref. [4,5]. It is analogous to the SPT concept in quantizing floating-point numbers.
- Most CORDIC-related algorithms can be explained using the AQ design framework

Reference

1. *Y.H. Hu,,* “CORDIC-based VLSI architectures for digital signal processing”, *IEEE Signal Processing Magazine* , Vol. 9, Issue: 3, pp. 16 -35, July 1992.
2. *E. Antelo, J. Villalba, J.D. Bruguera, and E.L. Zapata,”* High performance rotation architectures based on the radix-4 CORDIC algorithm”, *IEEE Transactions on Computers*, Vol. 46, Issue: 8, Aug. 1997.
3. C.S Wu and A.Y. Wu, “Modified vector rotational CORDIC (MVR-CORDIC) algorithm and architecture”, *IEEE Trans. Circuits and systems-II: analog and digital signal processing*, vol. 48, No. 6, June 2001.
4. A.Y. Wu and C.S. Wu, “A unified design framework for vector rotational CORDIC family based on angle quantization process”, *In Proc. of 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. 2, pp. 1233 -1236.
5. A. Y. Wu and Cheng-Shing Wu, “A Unified View for Vector Rotational CORDIC Algorithms and Architectures based on Angle Quantization Approach,” in *IEEE Trans. Circuits and Systems Part-I: Fundamental Theory and Applications*, vol. 49, no. 10, pp. 1442-1456, Oct. 2002.
6. C. S. Wu and A. Y. Wu, “A novel trellis-based searching scheme for EEAS-based CORDIC algorithm” In Proc. of 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing, Vol. 2 , pp. 1229 -1232.