Real Time Digital Signal Processing

UTN-FRBA
2010

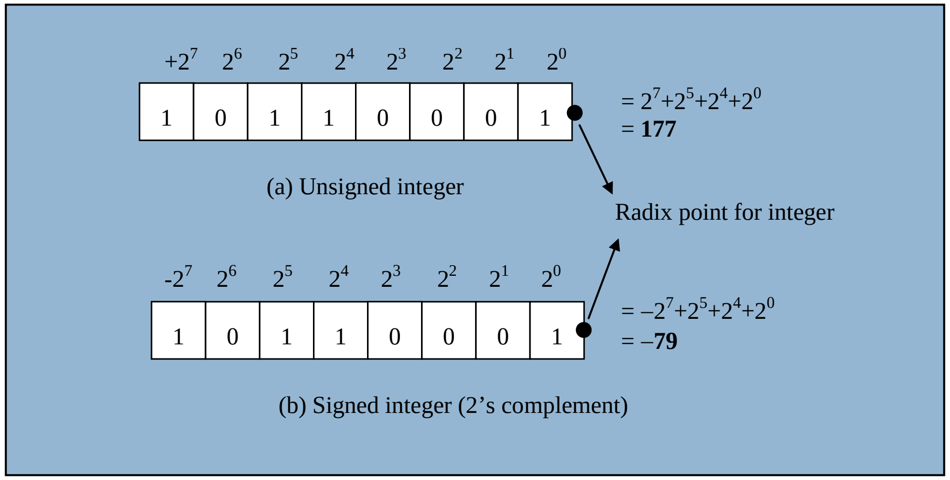
Eng. Julian Bruno

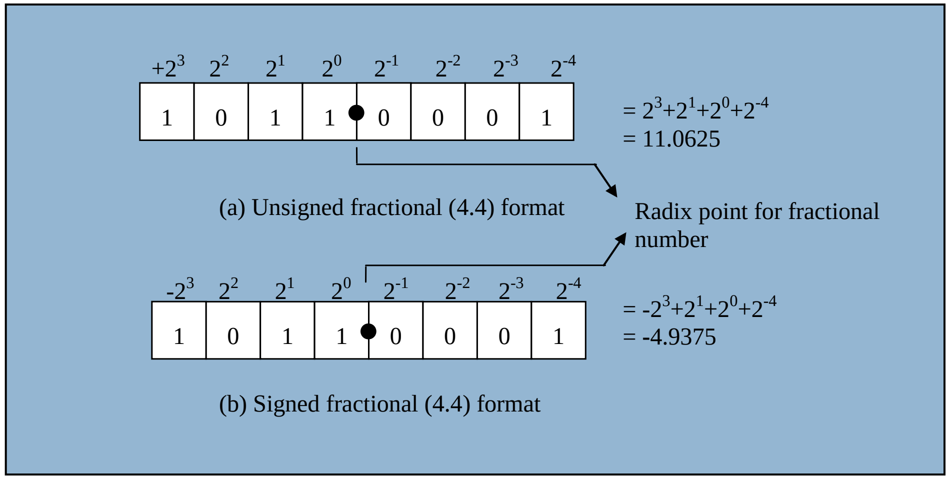
DSP fundamentals

Number representation and word-length effects.

Number Representation

- 8-bit Binary Data Format (Integer)
- 8-bit Binary Data Format (Fractional)





Integer Fixed-Point Representation

- N-bit fixed point, 2's complement integer representation

$X = -b_{N-1}\, 2^{N-1} + b_{N-2}\, 2^{N-2} + \ldots + b_0 2^0$

- Difficult to use due to possible overflow
  - In a 16-bit processor, the dynamic range is -32,768 to 32,767.
  - Example:

    200 × 350 = 70000, which is an overflow!

Fractional Fixed-Point Representation

- Also called Q-format
- Fractional representation suitable for DSPs algorithms.
- Fractional number range is between 1 and -1
- Multiplying a fraction by a fraction always results in a fraction and will not produce an overflow (e.g., 0.99 x 0.9999 less than 1)
- Successive additions may cause overflow
- Represent numbers between
  - -1.0 and $1 - 2^{-(N-1)}$, when N is number of bits

$$x_{10} = -b_N + \sum_{m=0}^{N-1} b_m . 2^{m-N}$$

Decimal equivalency for QN formats

General Fixed-Point Representation

- Qm.n notation
  - m bits for integer portion
  - n bits for fractional portion
  - Total number of bits N = m + n + 1, for signed numbers
  - Example: 16-bit number (N=16) and Q2.13 format
    - 2 bits for integer portion
    - 13 bits for fractional portion
    - 1 signed bit (MSB)
  - Special cases:
    - 16-bit integer number (N=16) => Q15.0 format
    - 16-bit fractional number (N = 16) => Q0.15 format; also known as Q.15 or Q15

$$\boxed{x_{10} = -b_{N-1} 2^{m} + \sum_{l=0}^{N-2} b_l 2^{l-n}}$$

Decimal equivalency for QM.N formats

Dynamic Ranges and Precision

Format (N.M)		Largest positive value (0x7FFF)	Least negative value (0x8000)	Precision (0x0001)		DR(dB)
1	15	0,999969482421875	-1	3,05176E-05	2^-15	90,30873362
2	14	1,99993896484375	-2	6,10352E-05	2^-14	90,30873362
3	13	3,9998779296875	-4	0,00012207	2^-13	90,30873362
4	12	7,999755859375	-8	0,000244141	2^-12	90,30873362
5	11	15,99951171875	-16	0,000488281	2^-11	90,30873362
6	10	31,99902344	-32	0,000976563	2^-10	90,30873362
7	9	63,99804688	-64	0,001953125	2^-9	90,30873362
8	8	127,9960938	-128	0,00390625	2^-8	90,30873362
9	7	255,9921875	-256	0,0078125	2^-7	90,30873362
10	6	511,984375	-512	0,015625	2^-6	90,30873362
11	5	1023,96875	-1024	0,03125	2^-5	90,30873362
12	4	2047,9375	-2048	0,0625	2^-4	90,30873362
13	3	4095,875	-4096	0,125	2^-3	90,30873362
14	2	8191,75	-8192	0,25	2^-2	90,30873362
15	1	16383,5	-16384	0,5	2^-1	90,30873362
16	0	32767	-32768	1	2^-0	90,30873362

Scale Factors and Dynamic Range

Format	Scaling factor ( )	Range in Hex (fractional value)
(1.15)	$2^{15} = 32768$	0x7FFF (0.99) $\rightarrow$ 0x8000 (–1)
(2.14)	$2^{14} = 16384$	0x7FFF (1.99) $\rightarrow$ 0x8000 (–2)
(3.13)	$2^{13} = 8192$	0x7FFF (3.99) $\rightarrow$ 0x8000 (–4)
(4.12)	$2^{12} = 4096$	0x7FFF (7.99) $\rightarrow$ 0x8000 (–8)
(5.11)	$2^{11} = 2048$	0x7FFF (15.99) $\rightarrow$ 0x8000 (–16)
(6.10)	$2^{10} = 1024$	0x7FFF (31.99) $\rightarrow$ 0x8000 (–32)
(7.9)	$2^{9} = 512$	0x7FFF (63.99) $\rightarrow$ 0x8000 (–64)
(8.8)	$2^{8} = 256$	0x7FFF (127.99) $\rightarrow$ 0x8000 (–128)
(9.7)	$2^{7} = 128$	0x7FFF (511.99) $\rightarrow$ 0x8000 (–512)
(10.6)	$2^{6} = 64$	0x7FFF (1023.99) $\rightarrow$ 0x8000 (–1024)
(11.5)	$2^{5} = 32$	0x7FFF (2047.99) $\rightarrow$ 0x8000 (–2048)
(12.4)	$2^{4} = 16$	0x7FFF (4095.99) $\rightarrow$ 0x8000 (–4096)
(13.3)	$2^{3} = 8$	0x7FFF (4095.99) $\rightarrow$ 0x8000 (–4096)
(14.2)	$2^{2} = 4$	0x7FFF (8191.99) $\rightarrow$ 0x8000 (–8192)
(15.1)	$2^{1} = 2$	0x7FFF (16383.99) $\rightarrow$ 0x8000 (–16384)
(16.0)	$2^{0} = 1$(Integer)	0x7FFF (32767) $\rightarrow$ 0x8000h (–32768)

Examples

Hex. Number	(16.0) format	(4.12) format	(1.15) format
0x7FFF			
0x8000			
0x1234			
0xABCD			
0x5566			

Number	(1.15) format	(2.14) format	(8.8) format	(16.0) format
0.5				
1.55				
–1				
–2.0345				

How to convert fractional number into integer

- Conversion from fractional to integer value:
  - Step 1: normalize the decimal fractional number to the range determined by the desired Q format
  - Step 2: Multiply the normalized fractional number by 2^n
  - Step 3: Round the product to the nearest integer
  - Step 4: Write the decimal integer value in binary using N bits.
- Example:
  - Convert the value 3,5 into an integer value that can be recognized by a DSP assembler using the Q15 format:
    - 1) Normalize: 3,5/4 = 0,875;
    - 2) Scale: 0.875*2^15= 28.672;
    - 3) Round: 28.672

How to convert integer into fractional number

- Numbers and arithmetic results are stored in the DSP processor in integer form.
- Need to interpret as a fractional value depending on Q format
- Conversion of integer into a fractional number for Qm.n format:
  - Divide integer by scaling factor of Qm.n => divide by 2^n
- Example:
  - Which Q15 value does the integer number 2 represent? $2/2^{15}=2*2^{-15}=2^{-14}$

Two's complement system

B=2	
011	3
010	2
001	1
000	0
111	-1
110	-2
101	-3
100	-4

Range -2^b to (2^b-1)
For b+1 data bits

Sign bit 011= 3
101=-3

DR_{dB} : Dynamic Range in dB

$$DR_{dB} = 20 \cdot \log_{10}\left(\frac{\text{largest possible word value}}{\text{smallest possible word value}}\right)$$

$$DR_{dB} = 20 \cdot \log_{10}\left(\frac{2^b-1}{1}\right) = 20 \cdot \log_{10}(2^b - 1)$$

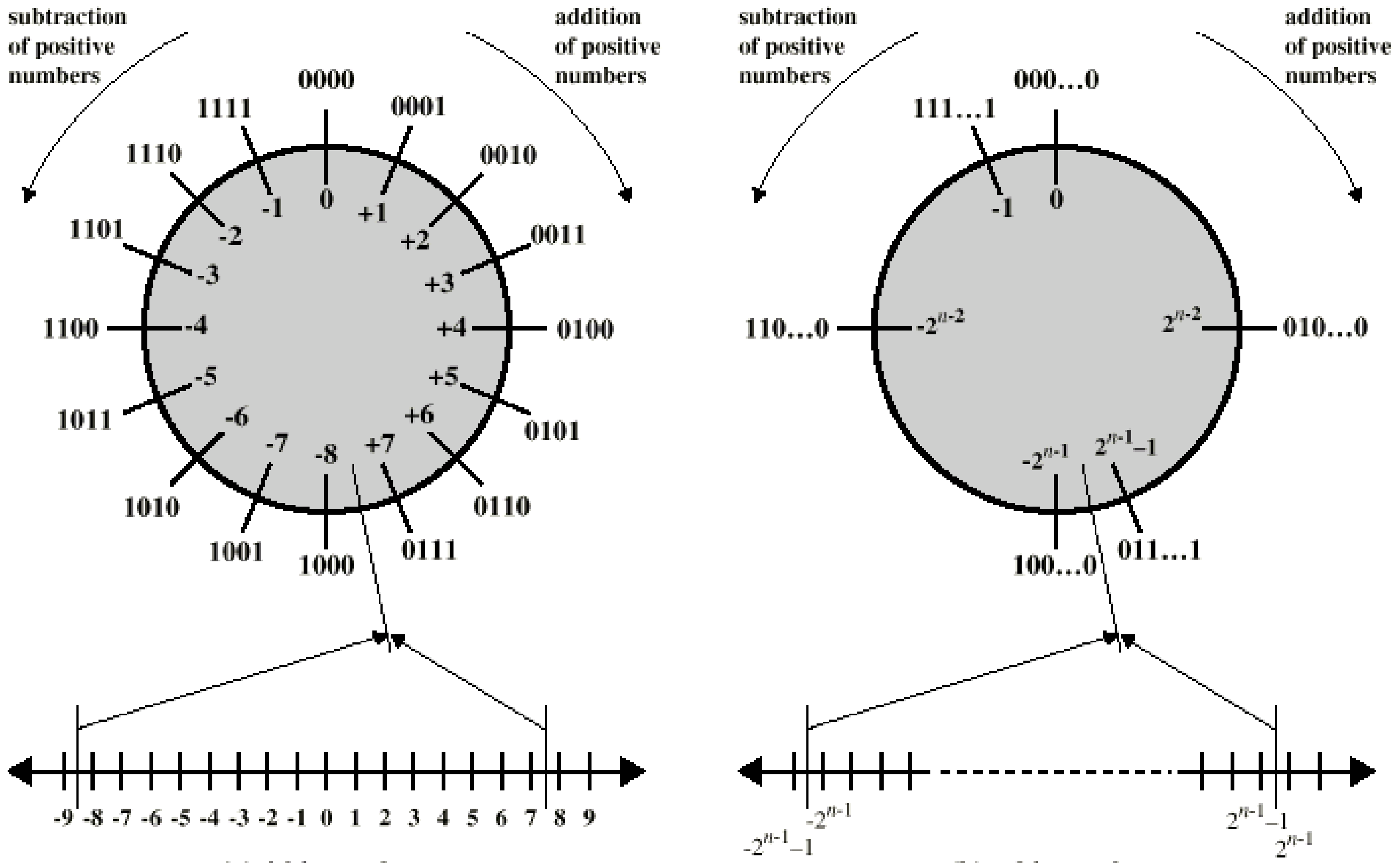
$$DR_{dB} = 20 \cdot \log_{10}(2^b) = 20 \cdot \log_{10}(2) \cdot b$$

$$DR_{dB} = 6.02 \cdot b\, dB$$

Negation mechanism	
Step	**Result**
Original number	011=3
1 complement	100
Add 1	101
	101=-3

- One bit for sign, B for number representation.
- Very popular system, widely used.
- Same logic for sum and subtraction.

Geometric Depiction of Twos Complement Integers



(a) 4-bit numbers

(b) n-bit numbers

Sum in Two's complement

Integer Format

	1	0	1	1	**-5_{10}**
+					
	0	1	1	0	**6_{10}**
	0	0	0	1	**1_{10}**

		1	0	1	1	**-5_{10}**
+						
		1	0	0	0	**-8_{10}**
	1	**0**	0	1	1	**-13_{10}**

Overflow !

Q3 Format

	1.	0	1	1	**-0.625_{10}**
+					
	0.	1	1	0	**0.75_{10}**
	0.	0	0	1	**0.125_{10}**

		1.	0	1	1	**-0.625_{10}**
+						
		1.	0	0	0	**-1_{10}**
	1	**0**.	0	1	1	**-1.625_{10}**

Overflow !

Sum in Two's complement

Different Formats

3.0	1 0 1 1 . 0	$\mathbf{-5_{10}}$	1.2	1 1 . 0 1 0	$\mathbf{-0.75_{10}}$
+			+		
1.1	0 0 0 1 . 1	$\mathbf{1.5_{10}}$	0.3	0 0 . 0 1 1	$\mathbf{0.375_{10}}$
3.1	1 1 0 0 . 1	$\mathbf{-3.5_{10}}$	1.3	1 1 . 1 0 1	$\mathbf{-0.375_{10}}$

For C=A+B, where

A is in P.Q format

B is in R.S format

The result C is in max(P,R).max(Q,S) format

Multiplication in Two's complement

Integer Format

4.0					1	0	1	1.	-5_{10}
x									
4.0					0	1	1	0.	6_{10}
					0	0	0	0	
		1	1	1	0	1	1		
		1	1	0	1	1			
		0	0	0	0				
	1	1	1	0	0	0	1	0.	-30_{10}

Sign extension

Fractional Format Q3

1.3					1.	0	1	1	-0.625_{10}
x									
1.3					0.	1	1	0	0.75_{10}
					0	0	0	0	
		1	1	1	0	1	1		
		1	1	0	1	1			
		0	0	0	0				
	1	1.	1	0	0	0	1	0	-0.4688_{10}

For C=AxB, where

A and B are B bits wide, C is 2B bits wide.

Multiplication in Two’s complement

Different formats

		3.1		1	0	1.	1	$\mathbf{-2.5_{10}}$
		x						
		2.2		0	1.	1	0	$\mathbf{1.5_{10}}$
				0	0	0	0	
	1	1	1	0	1	1		
	1	1	0	1	1			
	0	0	0	0				
1	1	1	0	0.	0	1	0	$\mathbf{-3.75_{10}}$

Sign extension

			1.3	1.	0	1	1	$\mathbf{-0.625_{10}}$
			x					
			2.2	0	1.	1	0	$\mathbf{1.5_{10}}$
				0	0	0	0	
	1	1	1	0	1	1		
	1	1	0	1	1			
	0	0	0	0				
1	1	1.	0	0	0	1	0	$\mathbf{-0.9375_{10}}$

For C=AxB, where

A is in P.Q format and B is in R.S format

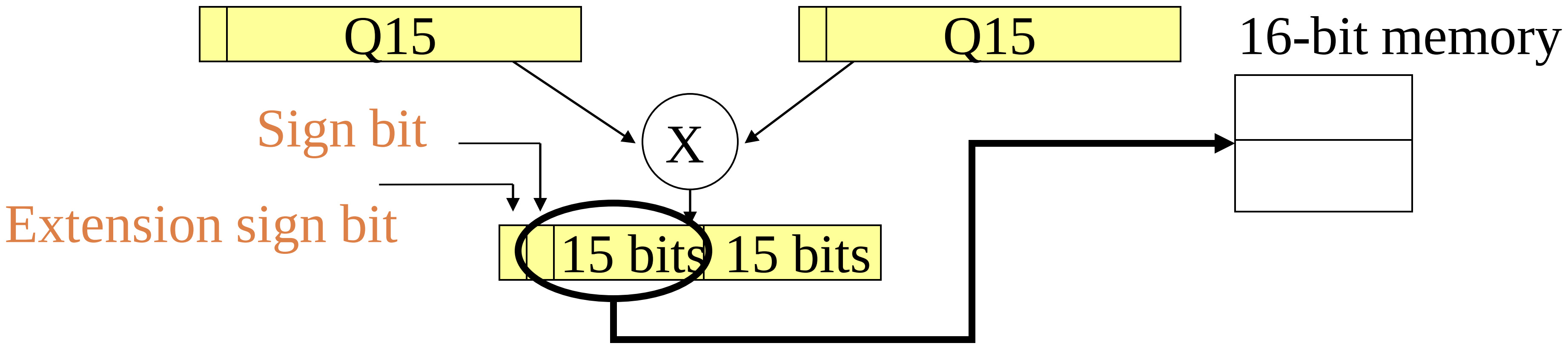
The result’s is in (P+R).(Q+S)

Multiplication: Why MSB is Redundant?

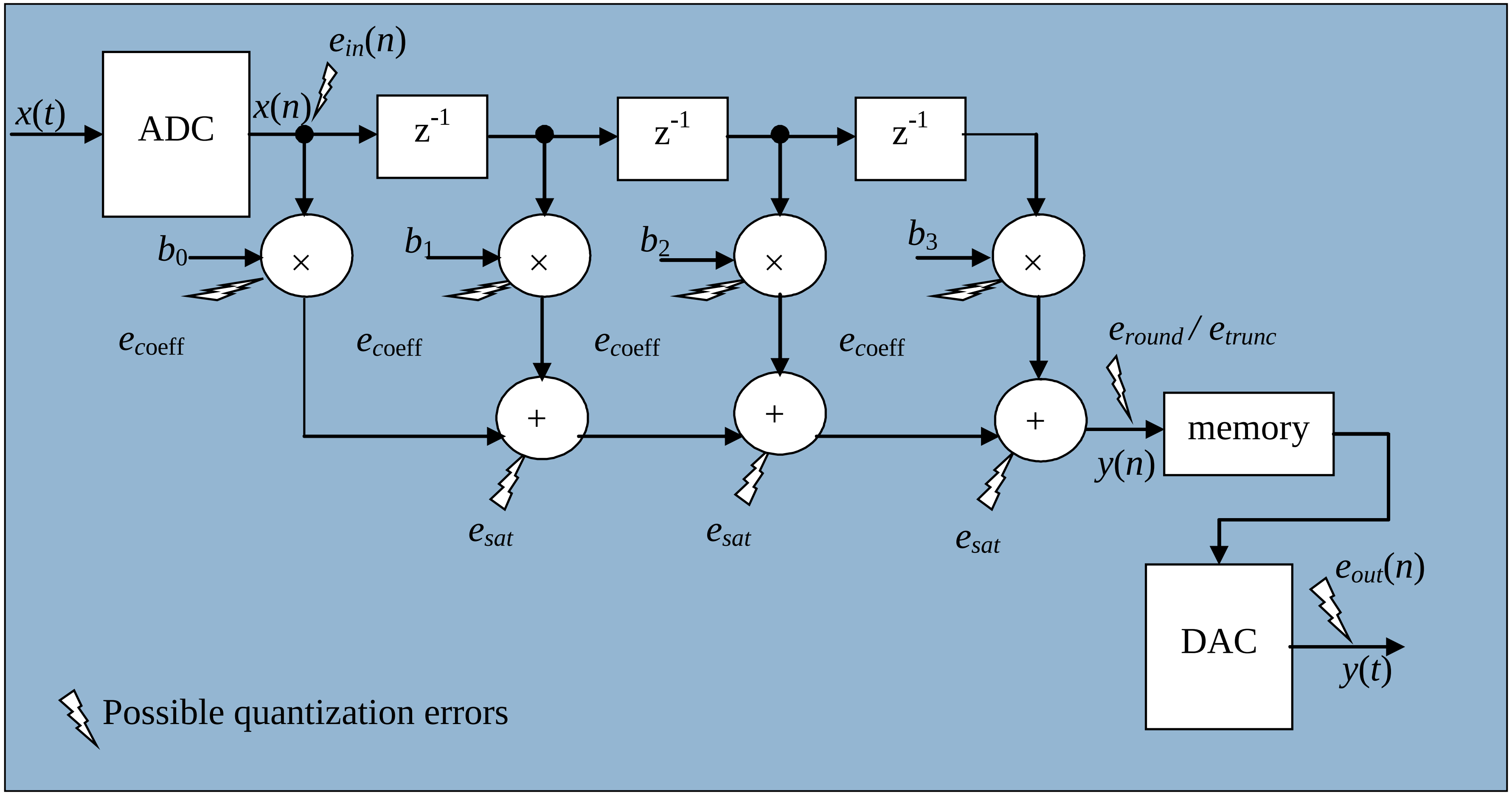
- Number represented by 4 bits (N = 4)
- 2’s complement range is from -8 to +7.
- The min/max number obtained from multiplication is -56/+64 $\Rightarrow$ 7 bits are enough to represent the result.
- NxN => 2N-1 bits
- The additional MSB is a “sign extension bit” and can be removed
- Another way to interpret it is that if converted to unsigned the multiplication result will be (N-1) + (N-1) + 1 sign bits giving 2N – 1.

Q format Multiplication

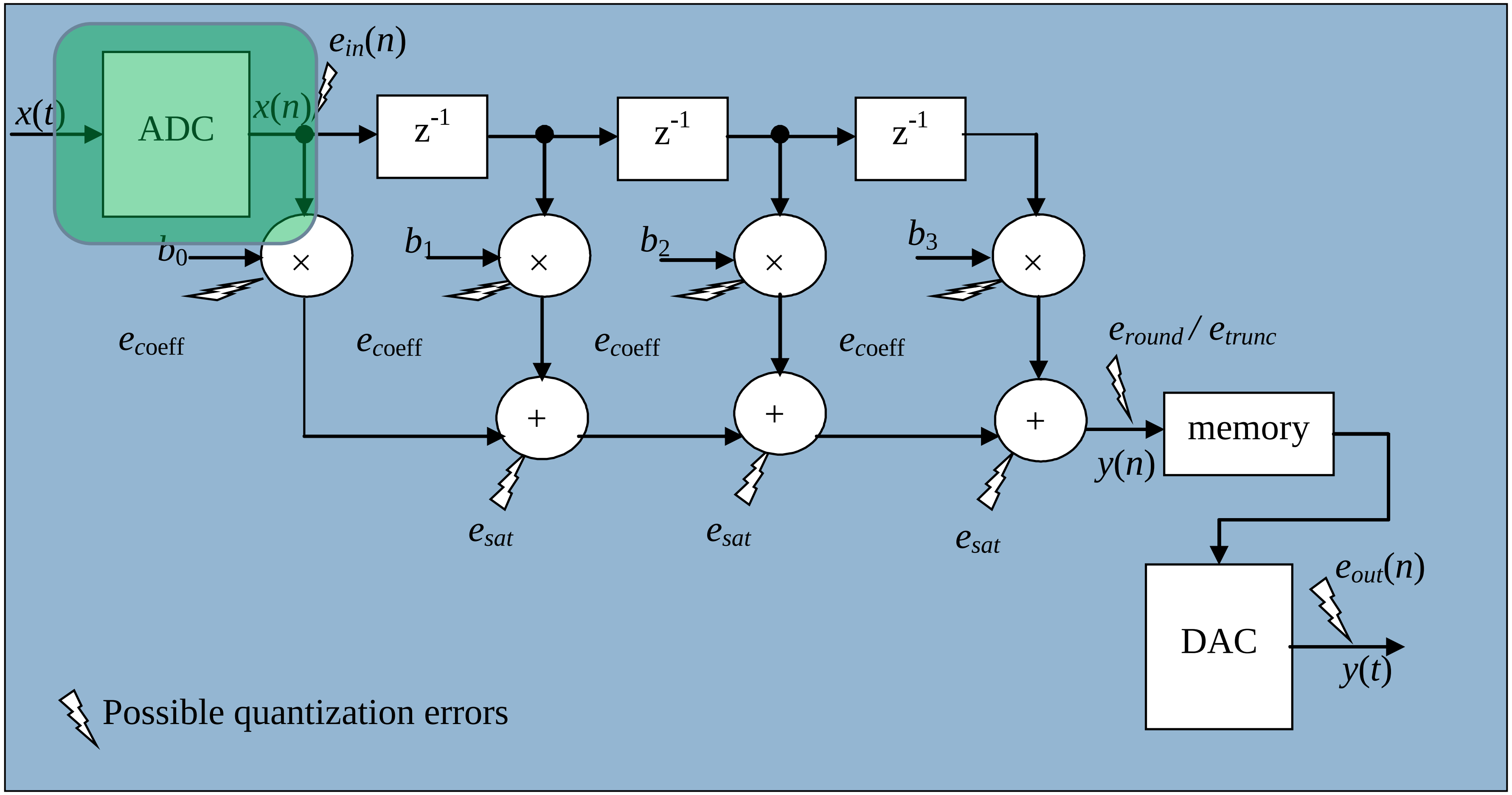
- Product of two Q15 numbers is Q30.
- So we must remember that the 32-bit product has two bits in front of the binary point.
- Since NxN multiplication yields 2N-1 result
- Addition MSB sign extension bit
- Typically, only the most significant 15 bits (plus the sign bit) are stored back into memory, so the write operation requires a ***left shift by one***.



Dynamic Range, Precision and Quantization errors

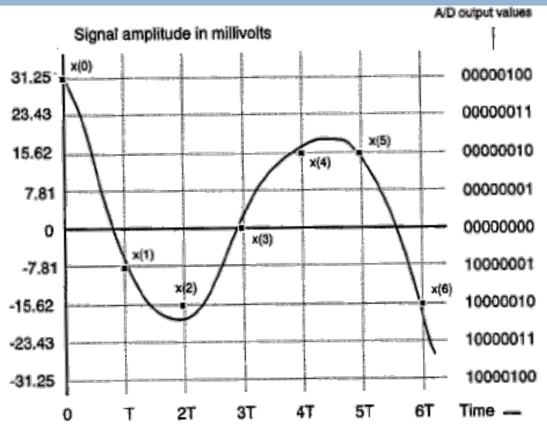


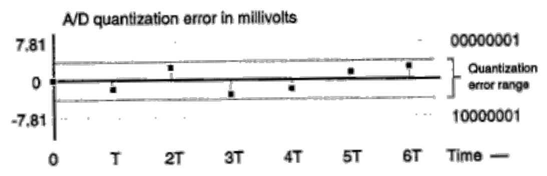
ADC errors and solutions



Analog Signal and Quantization

- The codec and system’s coefficients are the main generators of quantization noise.
- Codec’s noise can be thought as a uniformly distributed PDF between –LSB/2 and LSB/2.
- The SNR of an ADC is proportional to the word-length and the loading factor.
- Oversampling and Dithering





$$\boxed{\Delta = \frac{2Vp}{2^B}}$$

Quantization Step

$$\boxed{m_e = 0}$$

$$\boxed{\sigma_e^2 = \Delta^2/12}$$

Mean and Variance of Quantization Error

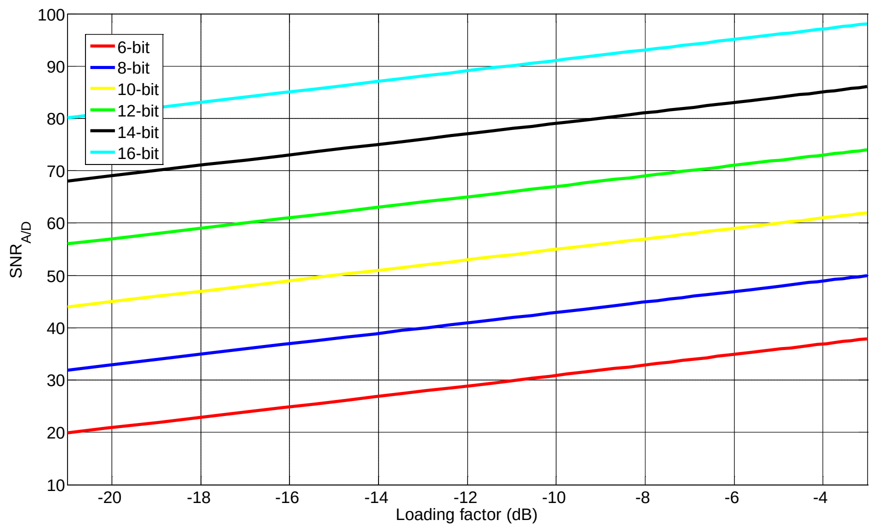
Analog Signal and Quantization

$$SNR_{A/D} = 10 \cdot \log_{10}\left(\frac{\text{input signal variance}}{\text{A/D quantization noise variance}}\right)$$

$$SNR_{A/D} = 10 \cdot \log_{10}\left(\frac{\sigma^2{}_{signal}}{\sigma^2{}_{A/D\,noise}}\right)$$

$$\sigma^2{}_{A/D\,noise} = \frac{\Delta^2}{12} = \frac{Vp^2}{3 \cdot 2^{2b}} \quad where \quad \Delta = \frac{2Vp}{2^b}$$

$$\sigma^2{}_{signal} = rms^2 = \left(\frac{Vp}{\sqrt{2}}\right)^2$$

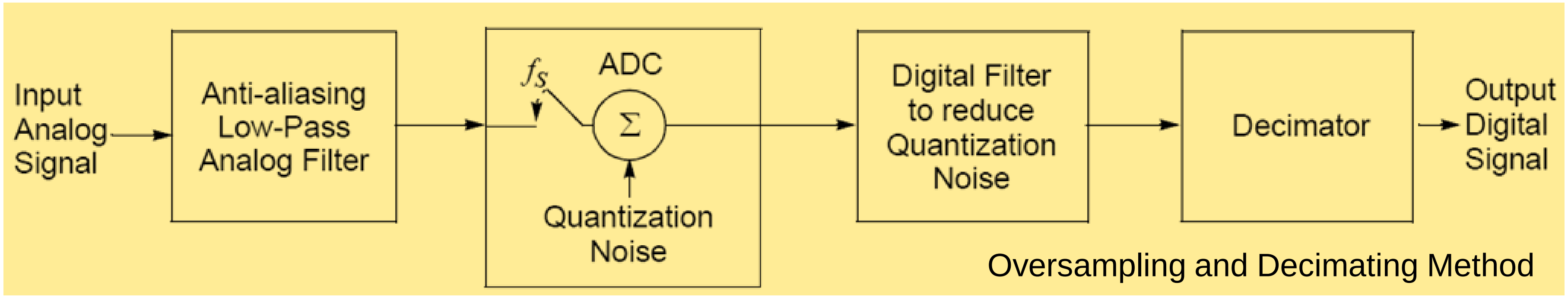


$$SNR_{A/D} = 10 \cdot \log_{10}\left(\frac{Vp^2/2}{Vp^2/3 \cdot 2^{2b}}\right) = 10 \cdot \log_{10}\left(1.5 \cdot 2^{2b}\right)$$

$$\boxed{SNR_{A/D} = 1.76dB + 6.02dB \cdot b} \Rightarrow \boxed{b = 16bits \Rightarrow SNR_{A/D} = 98.08dB}$$

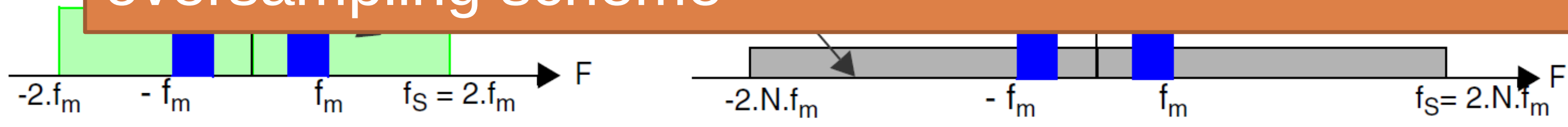
$$\text{Loading Factor}: LF = \frac{rms}{Vp} = \frac{\sigma_{signal}}{Vp} \quad ; \quad \sigma^2{}_{signal} = LF^2 Vp^2 \Rightarrow \boxed{SNR_{A/D} = 4.77dB + 6.02dB \cdot b + 20 \cdot \log_{10}(LF)}$$

Oversampling Method


Input Analog Signal
Anti-aliasing Low-Pass Analog Filter
f_S
ADC
Σ
Quantization Noise
Digital Filter to reduce Quantization Noise
Decimator
Output Digital Signal

Oversampling and Decimating Method

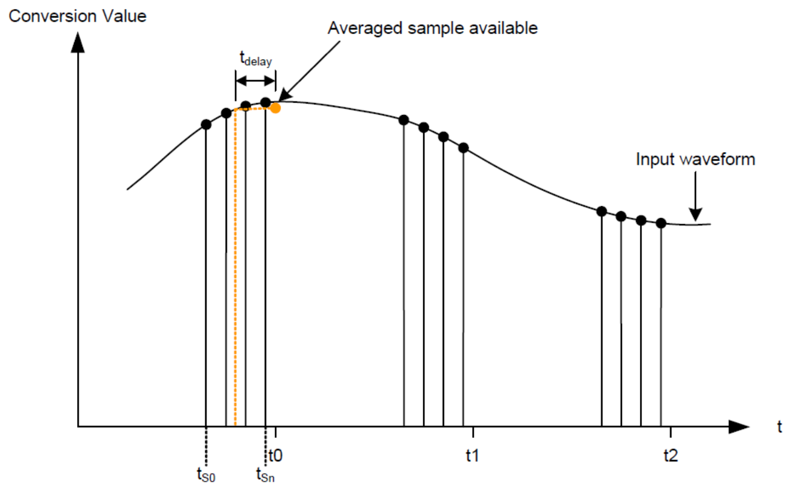
The number of bits used for the **lowpass filter's *coefficients*** and ***registers*** must exceed the original number of ADC bits in order to benefit from the oversampling scheme



$$\text{Prosessing Gain} : PG = 10\log\left(\frac{fs}{2BW}\right) \quad ; \quad \boxed{SNR_{A/D} = 1.76dB + 6.02dB \cdot b + 10\log\left(\frac{fs}{2BW}\right)}$$

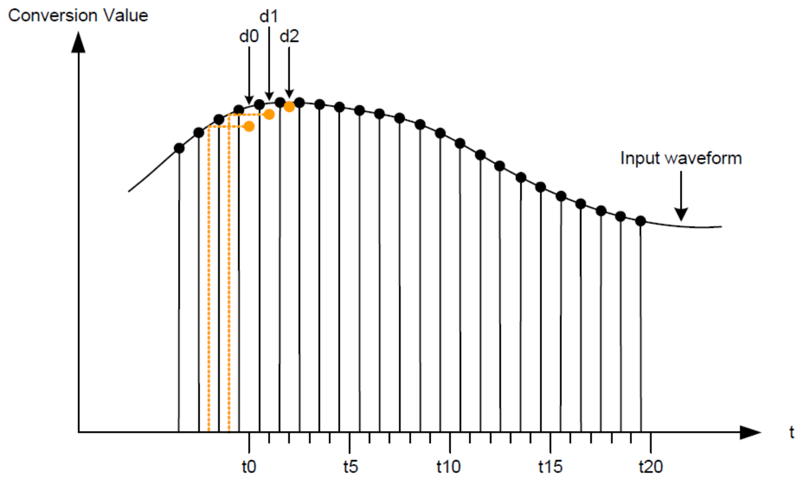
Oversampling Method

Normal Averaging



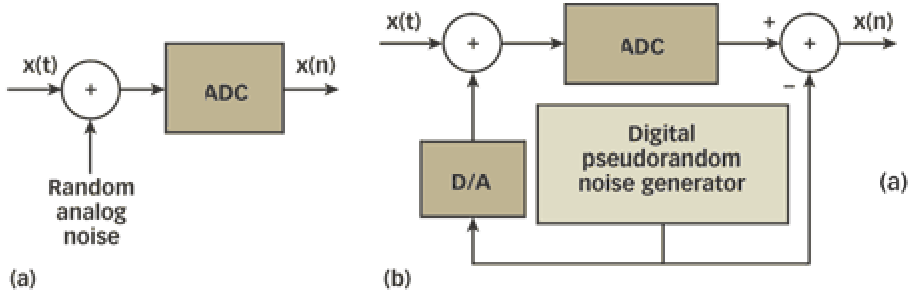
It's ideally used in cases where the sampling frequency is low compared to the sampling rate of the ADC

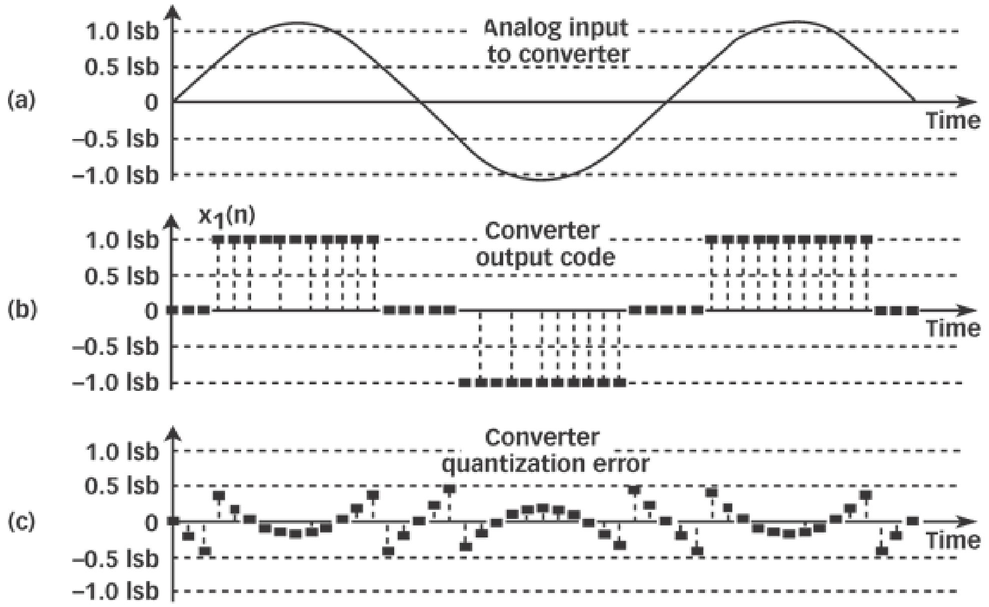
Rolling Average



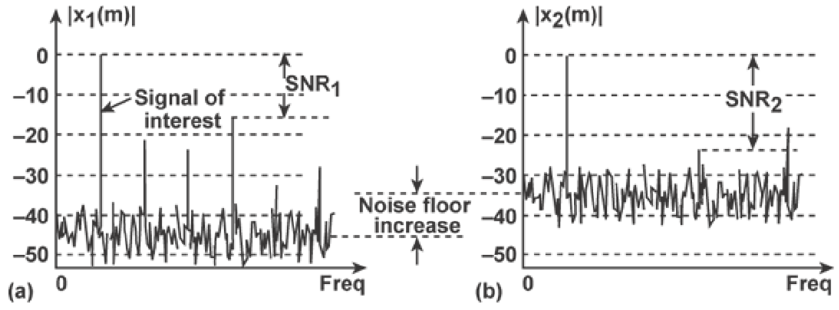
It's ideally suited for applications requiring oversampling and higher sample rates

Dithering Method





Dithering forces the quantization noise to lose its coherence with the original input signal.



- •Low-amplitude analog signals.
- •Highly periodic analog signals.
- •Slowly varying (DC to very low frequency) analog signals.

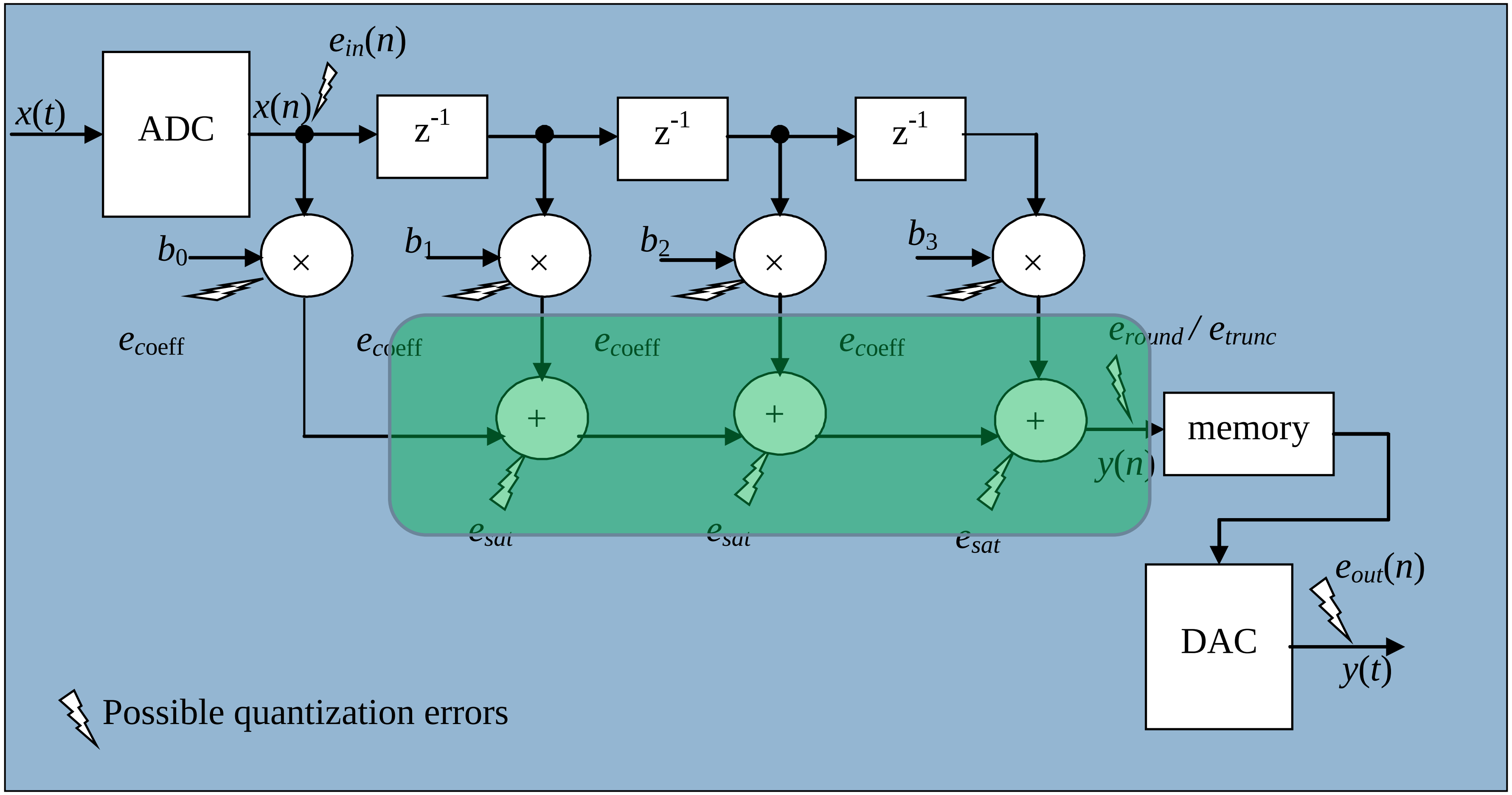
Analog Signal and Quantization

- $SNR_{A/D}$ >= SNR_{signal}
- In practice, $SNR_{A/D} = SNR_{A/D\ ideal}$ *-3 to 6 dB*
  - *Aperture jitter error*
  - *Missing output bit patterns*
  - *Other nonlinearities*
- It's imprudent to force an A/D convert's input to full scale. Use LF to determine A/D's SNR.
- Effective Numbers Of Bits (ENOB)

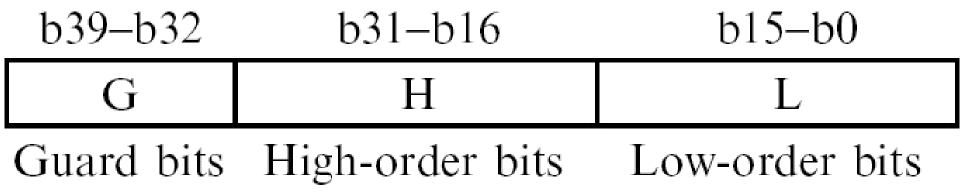
$$b_{eff} = \frac{SNR - 1.76}{6.02}$$

- SNR_{DSP} >= $SNR_{A/D}$

Overflow errors and solutions

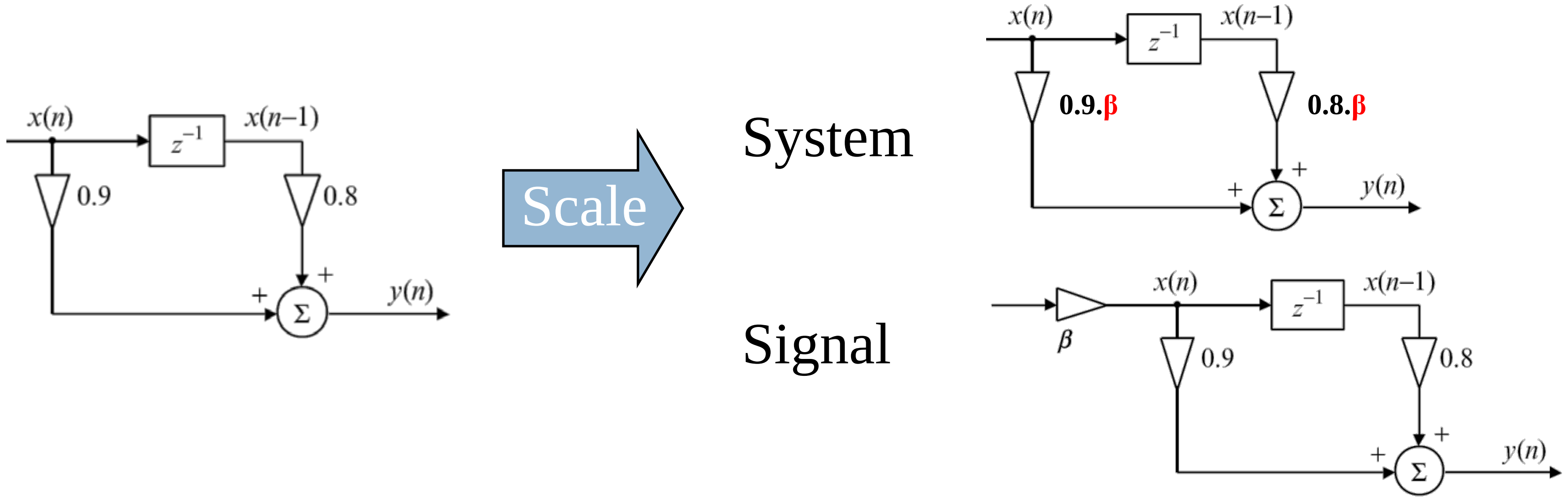


Avoiding overflow



- Always use the maximum capability (guard bits) of the accumulators during internal calculations.
- Only round (or truncate) the final results to the final data size and format if possible.
- There is (almost) no lost of precision when handling internal calculations with guard bits.

Avoiding overflow



- Scaling down a signal is the most effective technique to prevent overflow.
- Scaling down always implies loss of precision.
- Both scaling down and guard bits techniques must be used in order to avoid overflow.
- Always is more convenient to scale down system’s coefficients instead of signals.

Avoiding overflow

Effect of β in SNR

$$\mathrm{SNR} = 10\log_{10}\left(\frac{\beta^2\sigma_x^2}{\sigma_e^2}\right) = 4.77 + 6.02B + 10\log_{10}\sigma_x^2 + 20\log_{10}\beta.$$

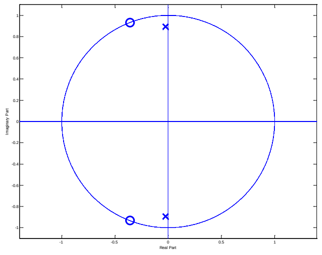
For example adopting β=0.5 implies a 6.02 dB decrease of SNR. This is equivalent that dividing by 2, rotating 1 time to the right, or losing 1 bit of resolution.

Never overflows

$$G < \frac{1}{x_{\max}\sum_{k=0}^{N-1}|h_k|}$$

- Scaling down always reduces SNR.
- It is possible to use an absolute safe or a more relaxed criteria to choose β value.
- Many times it is preferable to use different Q fractional formats within an algorithm.
- As overflow is very probable to happen in fixed point processors, special effort should be taken when coding algorithms and debugging.

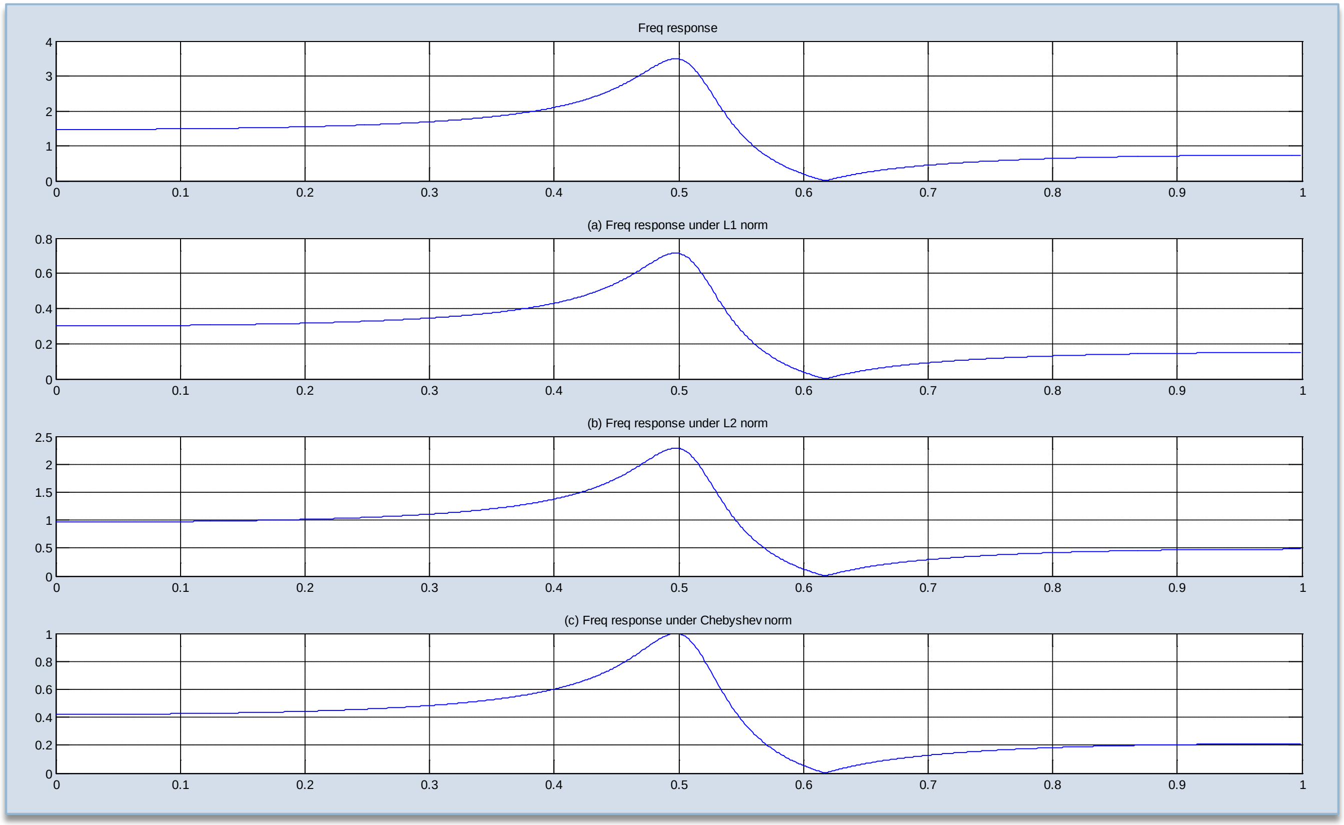
Avoiding overflow



$$G < \frac{1}{x_{\max} \sum_{k=0}^{N-1} |h_k|}$$

$$G < \frac{1}{x_{\max} \left(\sum_{k=0}^{N-1} h_k^2 \right)^{1/2}}$$

$$G < \frac{1}{x_{\max} \max[H(\omega_k)]}$$

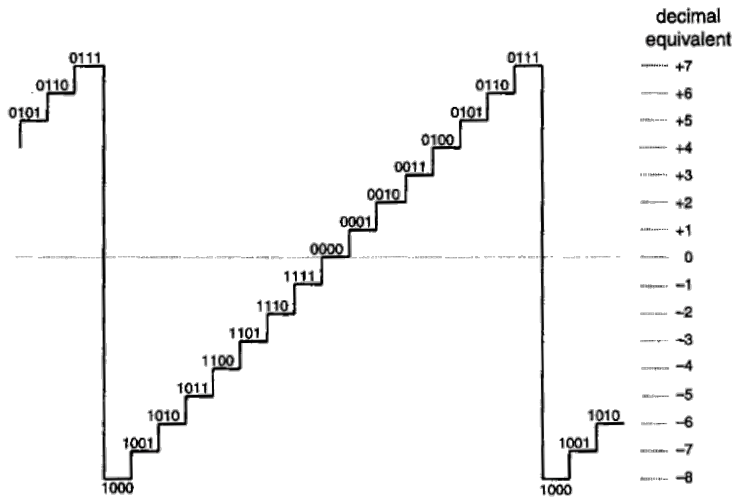


$$H(z) = \frac{1 + 0.72z^{-1} + z^{-2}}{1 + 0.052z^{-1} + 0.8z^{-2}}$$

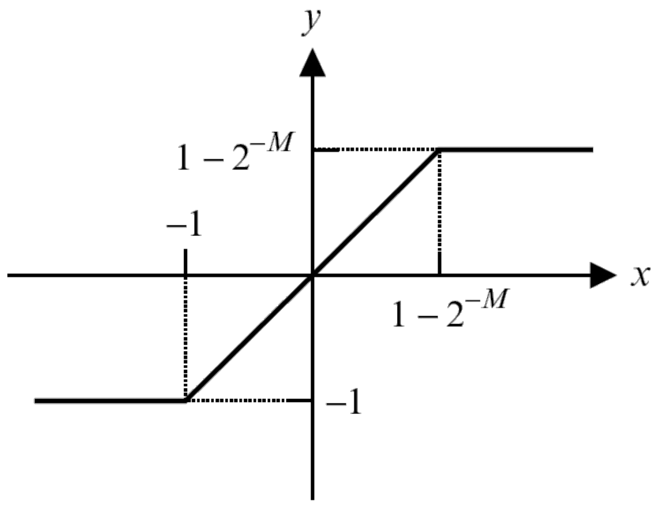
l1_norm = 4.8839
l2_norm = 1.5263
cheb_norm = 3.4926

Minimizing overflow effects

Without saturation arithmetic

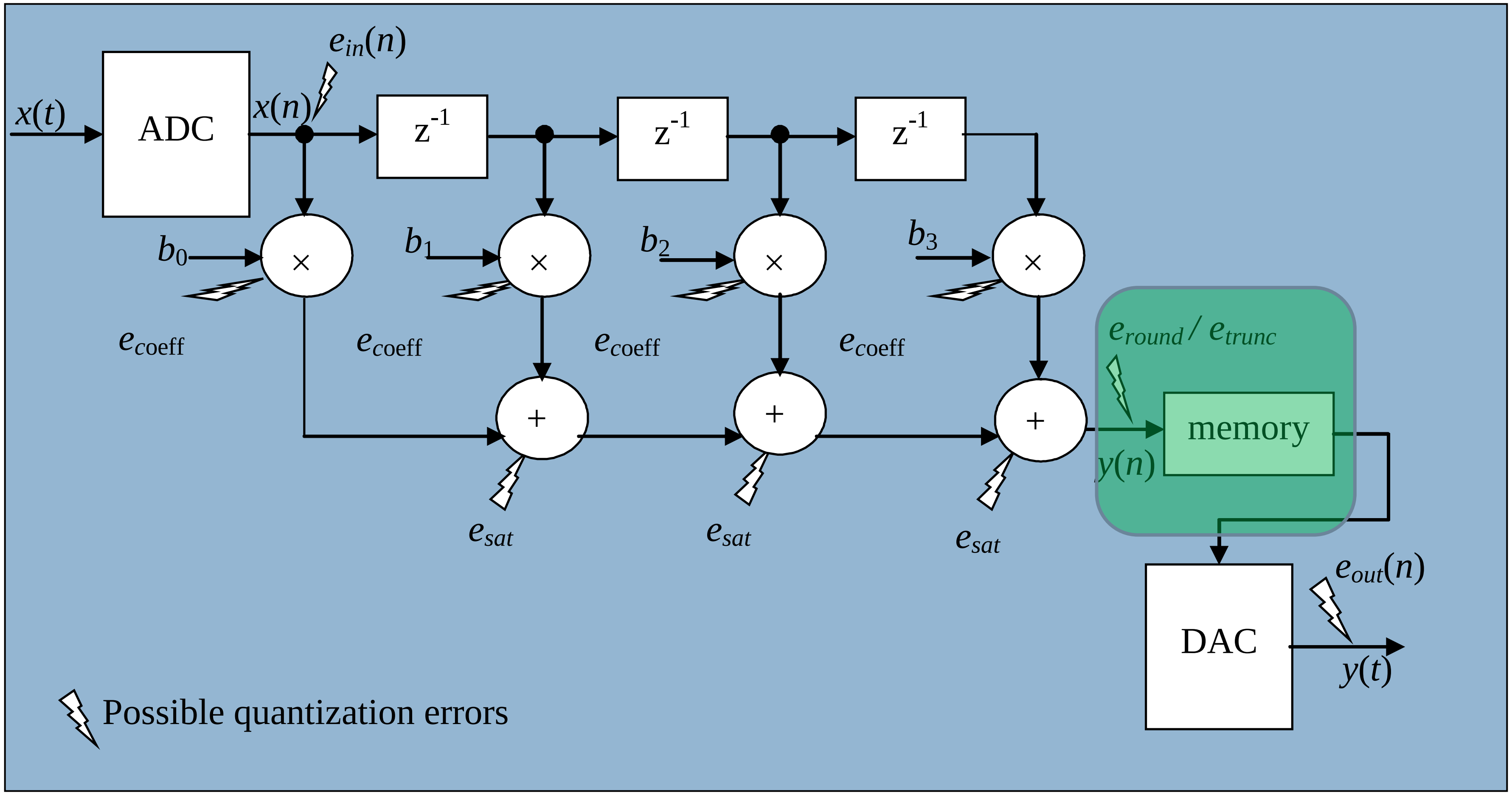


With saturation arithmetic

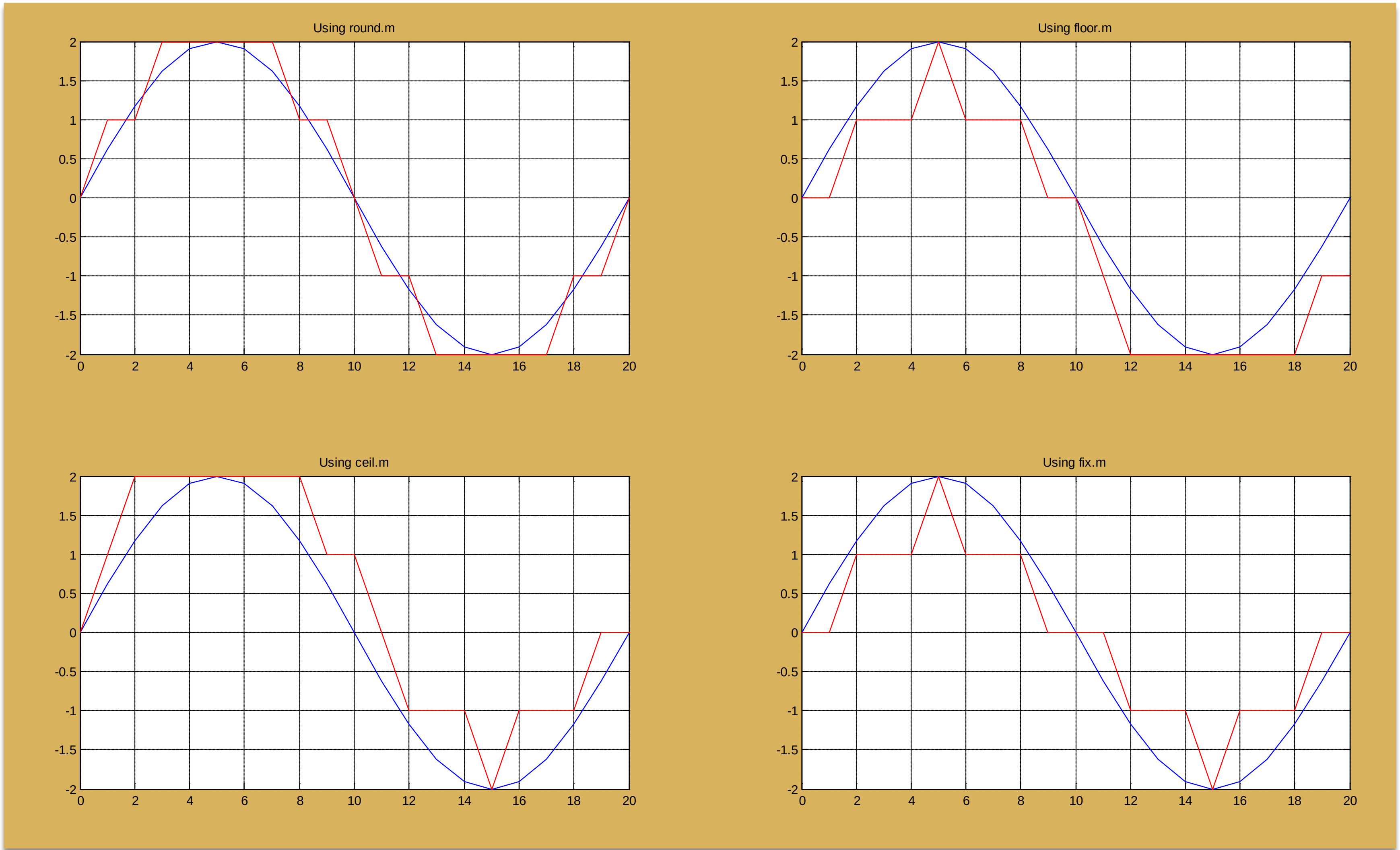


- Always use saturating arithmetic.
- In case overflow occurs, decrease the probability that an oscillation occurs.

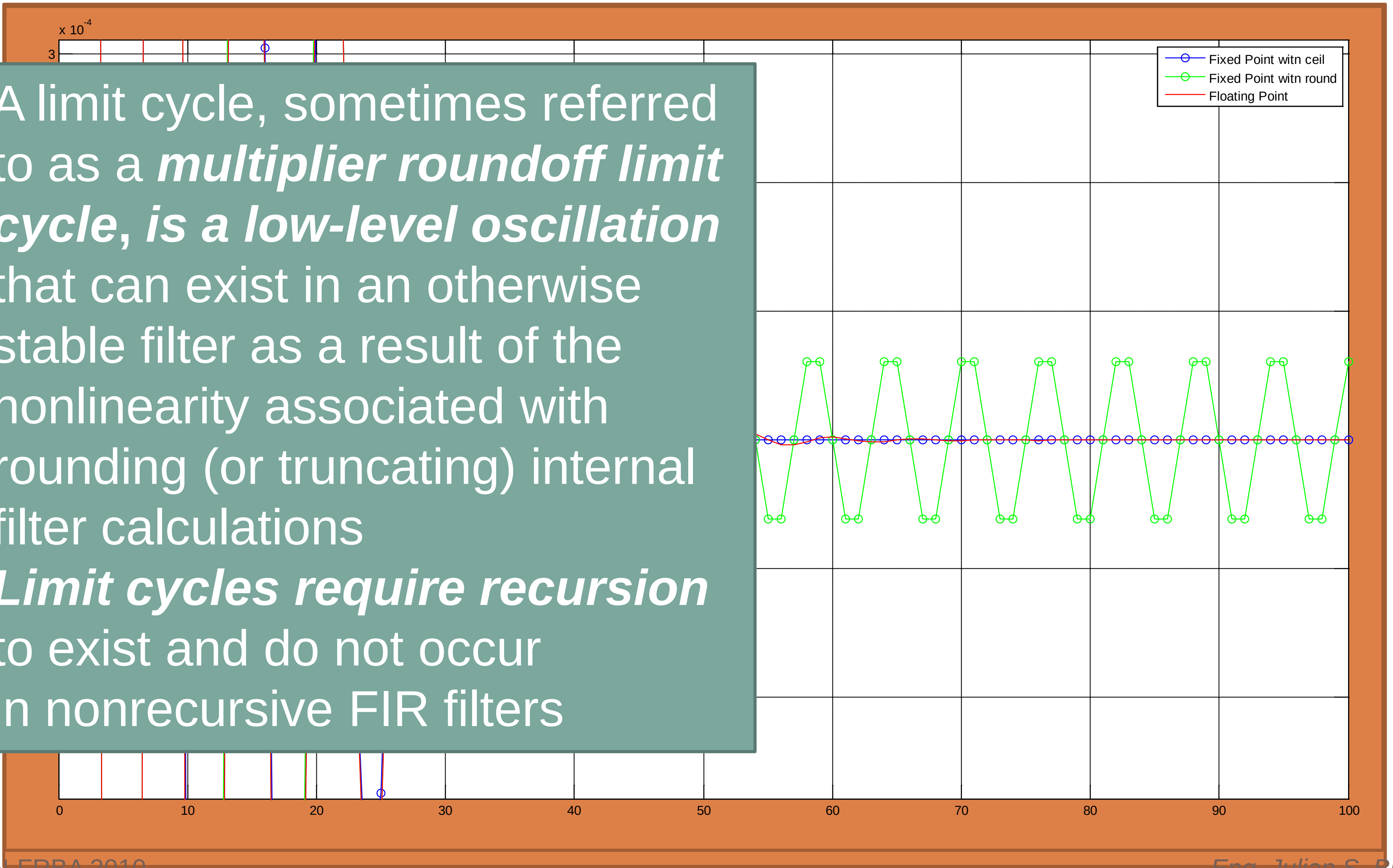
Truncation and Rounding



Truncation/Rounding in Multiply-Accumulate



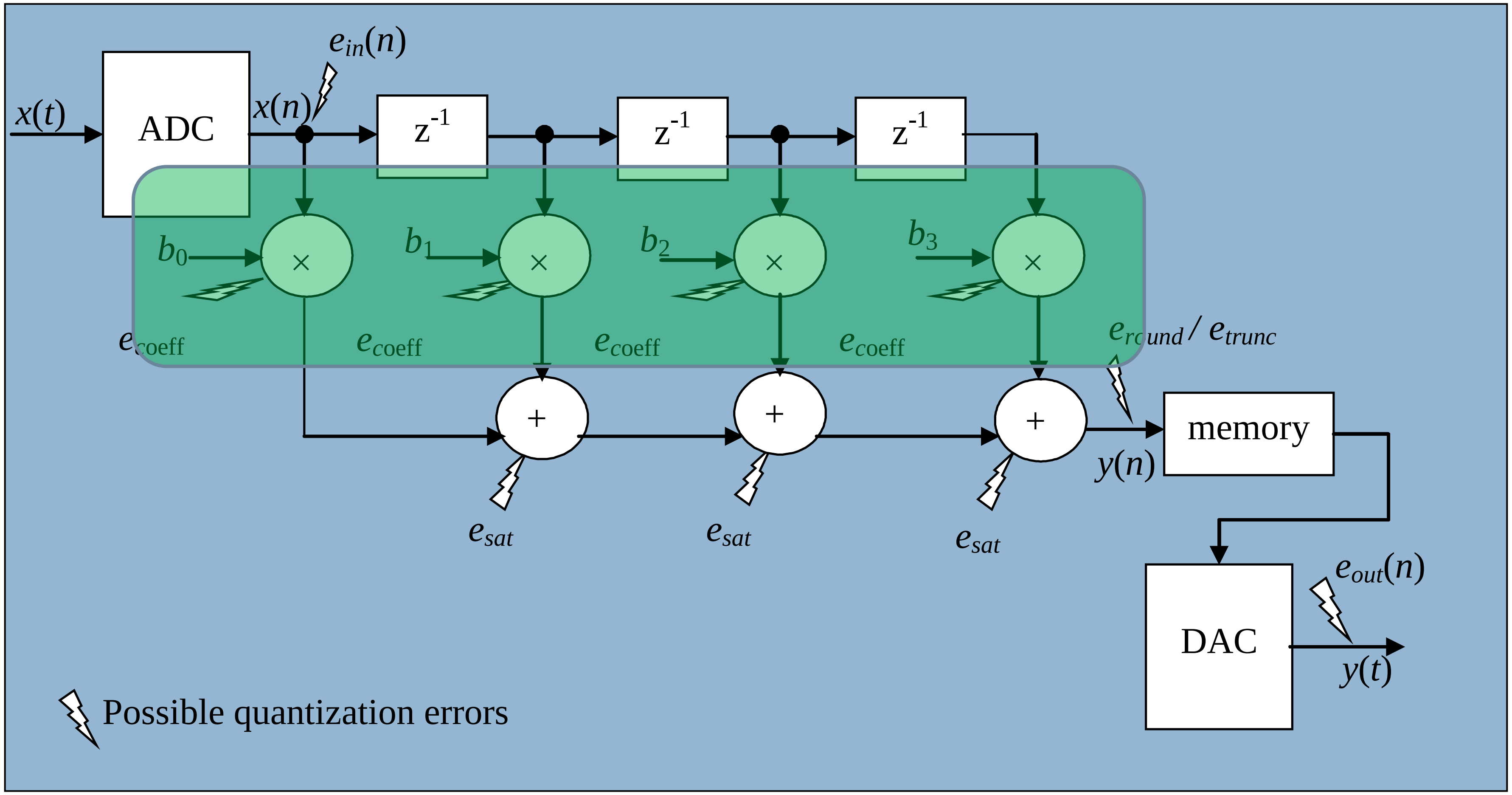
Truncation and Rounding



A limit cycle, sometimes referred to as a ***multiplier roundoff limit cycle*, *is a low-level oscillation*** that can exist in an otherwise stable filter as a result of the nonlinearity associated with rounding (or truncating) internal filter calculations

Limit cycles require recursion to exist and do not occur in nonrecursive FIR filters

Coefficient Quantization Error

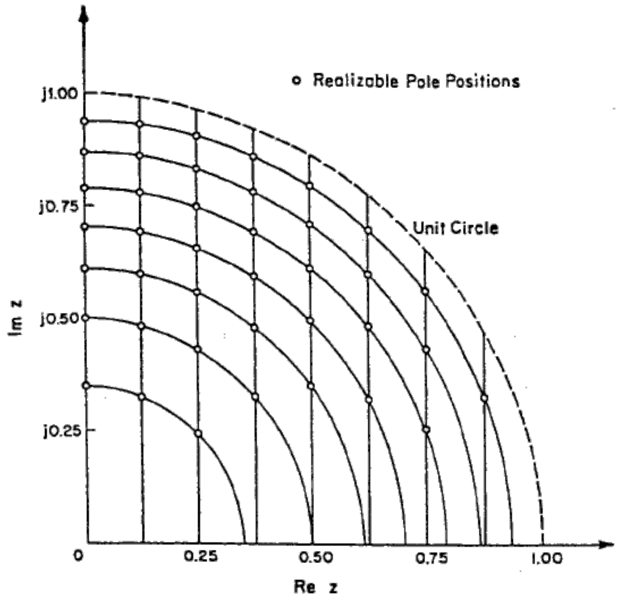


Quantization word-length effects

Complex conjugated two poles band pass

$$\begin{aligned} \lambda &= re^{\pm j\theta} \\ &= \lambda_r \pm j\lambda_i \\ &= r\cos(\theta) \pm jr\sin(\theta) \end{aligned}$$

$$H(z) = \frac{1}{1 - 2r\cos(\theta)z^{-1} + r^2 z^{-2}}$$

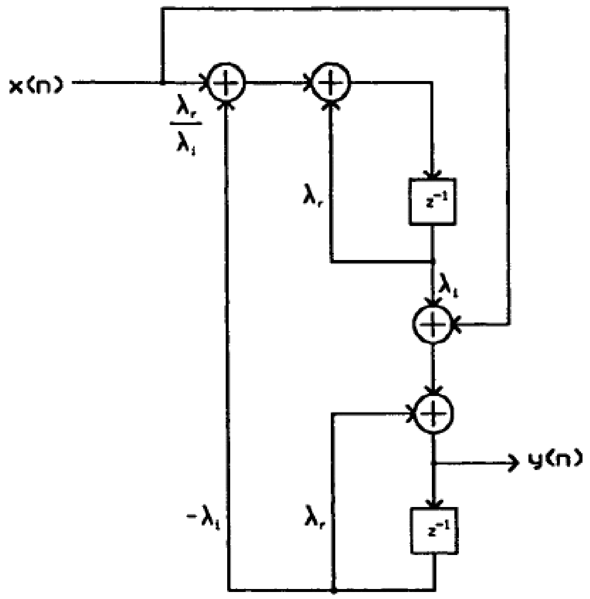


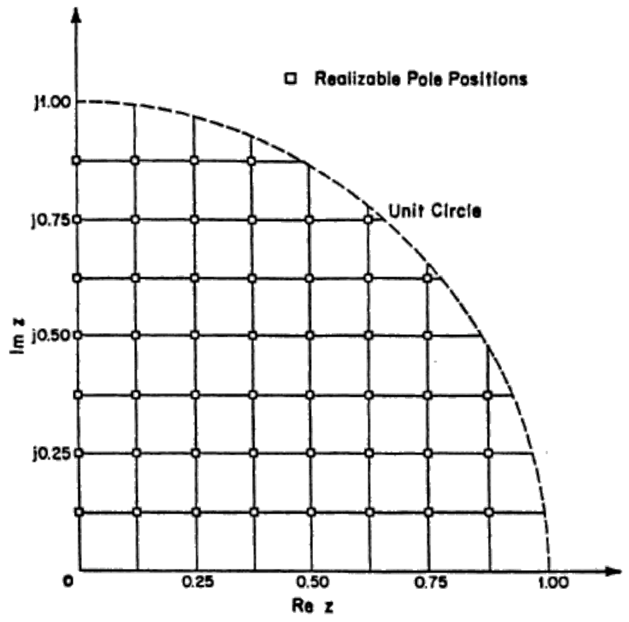
And its difference equation

$$y(n) = 2r\cos(\theta)y(n-1) - r^2 y(n-2) + x(n)$$

- When defining a system in term of its coefficients, the finite precision affect the behavior of the system itself.
- Though there is a grid of possible locations where system’s poles can be placed.
- This grid depends first of the word-length and second of the structure adopted to implement of the system.

Quantization word-length effects





- There are structures are less sensitive to coefficient quantization.
- There is a trade-off between efficiency and sensibility to coefficient quantization.

Finite Wordlength Effects (I)

- Discretization (quantization) of the filter coefficients has the effect of perturbing the location of the filter poles and zeroes. This deterministic frequency response error is referred to as **coefficient quantization error.**
- The use of finite precision arithmetic makes it necessary to quantize filter calculations by rounding or truncation. **Roundoff noise** is that error in the filter output that results from rounding or truncating calculations within the filter.

* Bruce W. Bomar - *University of Tennessee Space Institute*

Finite Wordlength Effects (II)

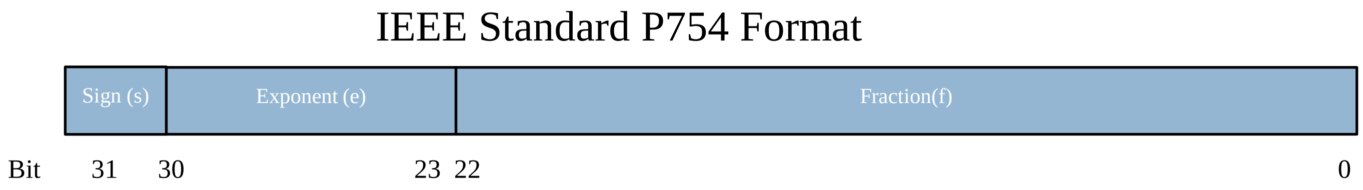
- Quantization of the filter calculations also renders the filter slightly nonlinear. However, for recursive filters with a zero or constant input, this nonlinearity can cause spurious oscillations called **limit cycles.**
- With fixed-point arithmetic it is possible for filter calculations to overflow. The term **overflow oscillation** refers to a high-level oscillation that can exist in an otherwise stable filter due to the nonlinearity associated with the overflow of internal filter calculations.

* Bruce W. Bomar - *University of Tennessee Space Institute*

Floating point representation

- This form of representation overcomes limitations of precision and dynamic range of fixed point.
- This format segment data in sign, exponent and mantissa.
- Mantissa is represented as a fixed point number.
- Exponent is represented in binary offset format.
- The greater the b_e the larger the dynamic range.
- The greater the b_m the larger the precision.
- There is a trade off between b_m and b_e, and the best balance occur at $b_e \approx b/4$ and $b_m \approx 3b/4$.
- DR = $6.02*2^{be}$

Floating point representation (I)



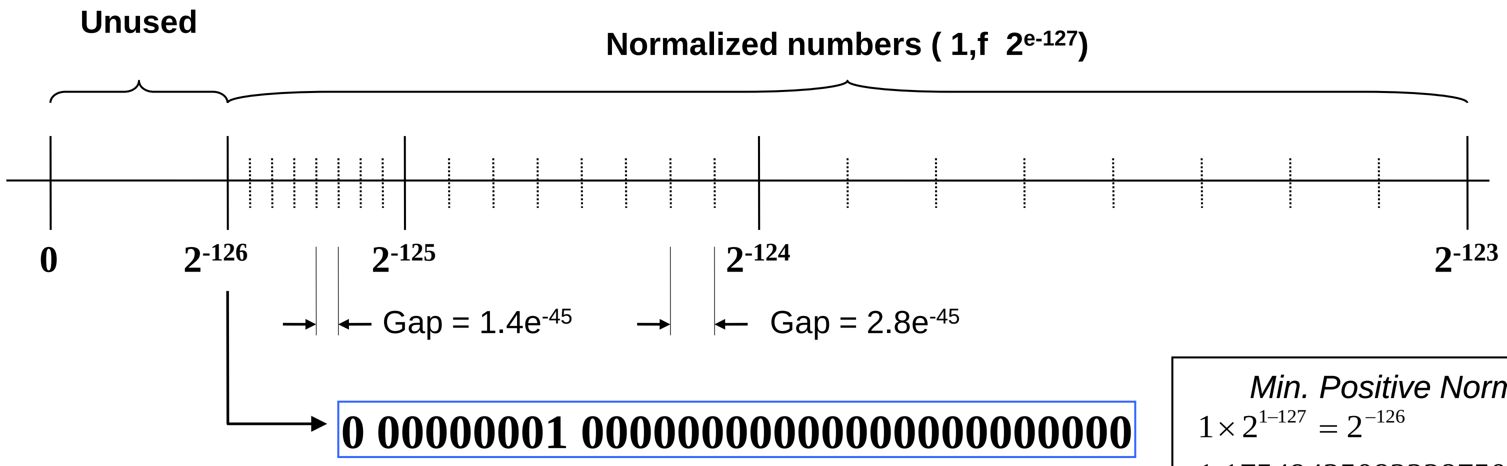
$$value_{ieee} = (-1)^s \cdot 1,f \cdot 2^{e-127}$$

- IEEE P754 is the most widely used floating point format.
- As the point is floating, a process called *normalization* is performed in order to use the full precision of b_m bits, while the exponent is adjusted properly.
- Floating point arithmetic usually requires lot of logical comparisons and branching, so software emulated floating achieves low performance
- Floating point DSPs implements in hardware all arithmetic handling, so these DSPs outperforms their fixed point counterparts in ease of use and performance (of course being more expensive too).

Floating point representation (II)

	Single Precision (32 bits)				Double Precision (64 bits)			
	Sign	Biased exponent	Fraction	Value	Sign	Biased exponent	Fraction	Value
Positive zero	0	0	0	0	0	0	0	0
Negative zero	1	0	0	-0	1	0	0	-0
Plus infinity	0	255(all 1s)	0	∞	0	255(all 1s)	0	∞
Minus infinity	1	255(all 1s)	0	$-\infty$	1	255(all 1s)	0	$-\infty$
NaN	0 or 1	255(all 1s)	$\neq 0$	NaN	0 o 1	255(all 1s)	$\neq 0$	NaN
Positive normalized	0	$0 < e < 255$	f	2^{e-127} (1,f)	0	$0 < e < 2047$	f	2^{e-1023} (1,f)
Negative normalized	1	$0 < e < 255$	f	-2^{e-127} (1,f)	1	$0 < e <2047$	f	-2^{e-1023} (1,f)
Positive denormalized	0	0	$f \neq 0$	2^{-126} (0,f)	0	0	$f \neq 0$	2^{-1022} (0,f)
Negative denormalized	1	0	$f \neq 0$	-2^{-126} (0,f)	1	0	$f \neq 0$	-2^{-1022} (0,f)

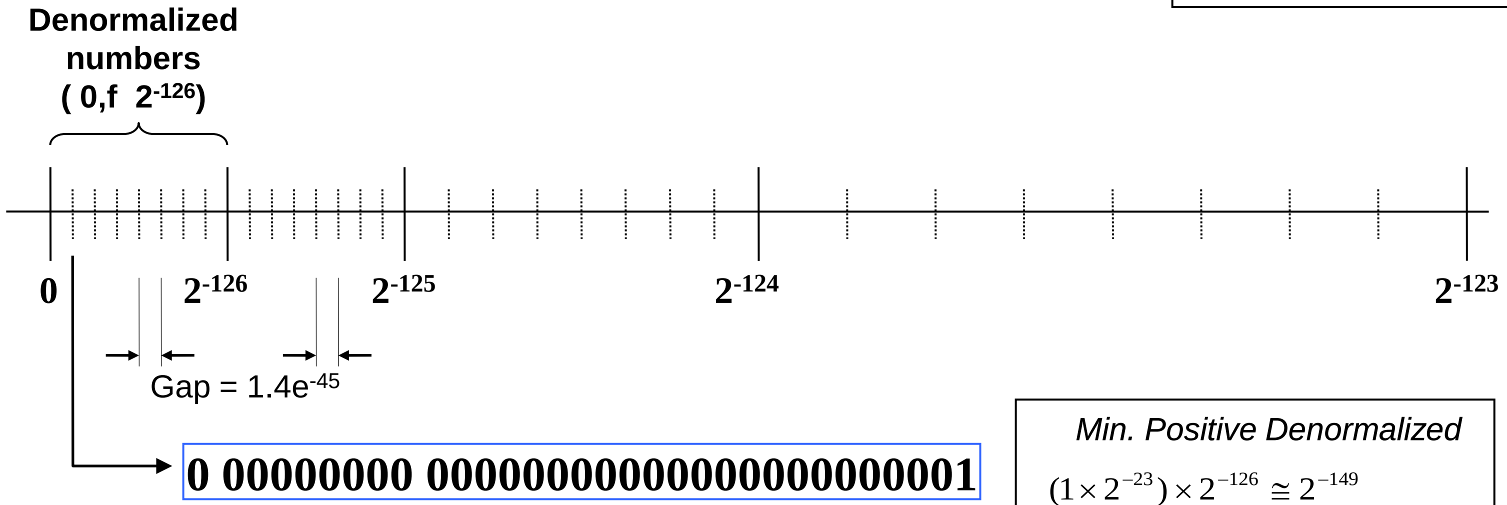
Normalized & Denormalized numbers (32-bit format)



Min. Positive Normalized

$1 \times 2^{1-127} = 2^{-126}$

1.17549435082287507968e - 38

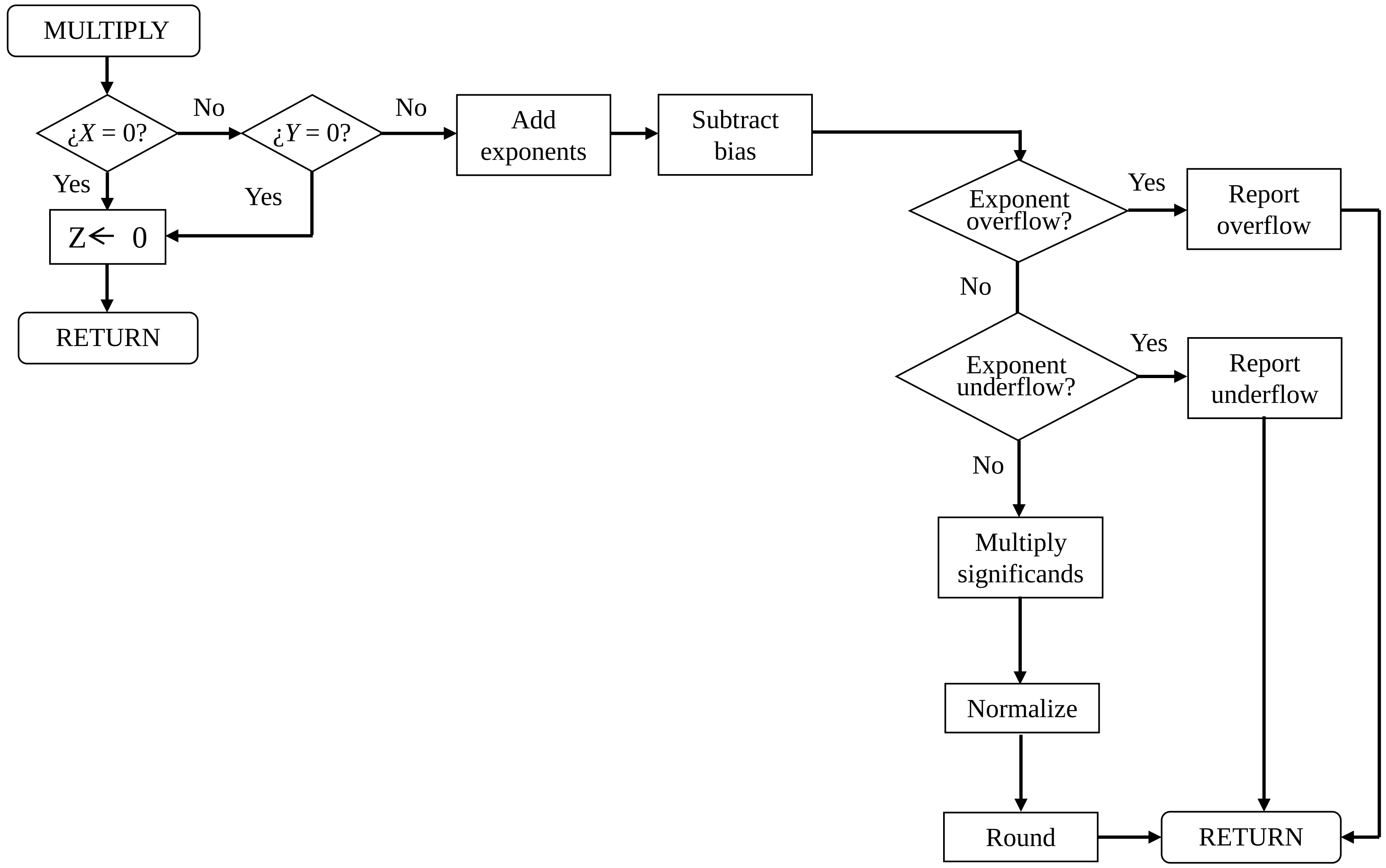


Min. Positive Denormalized

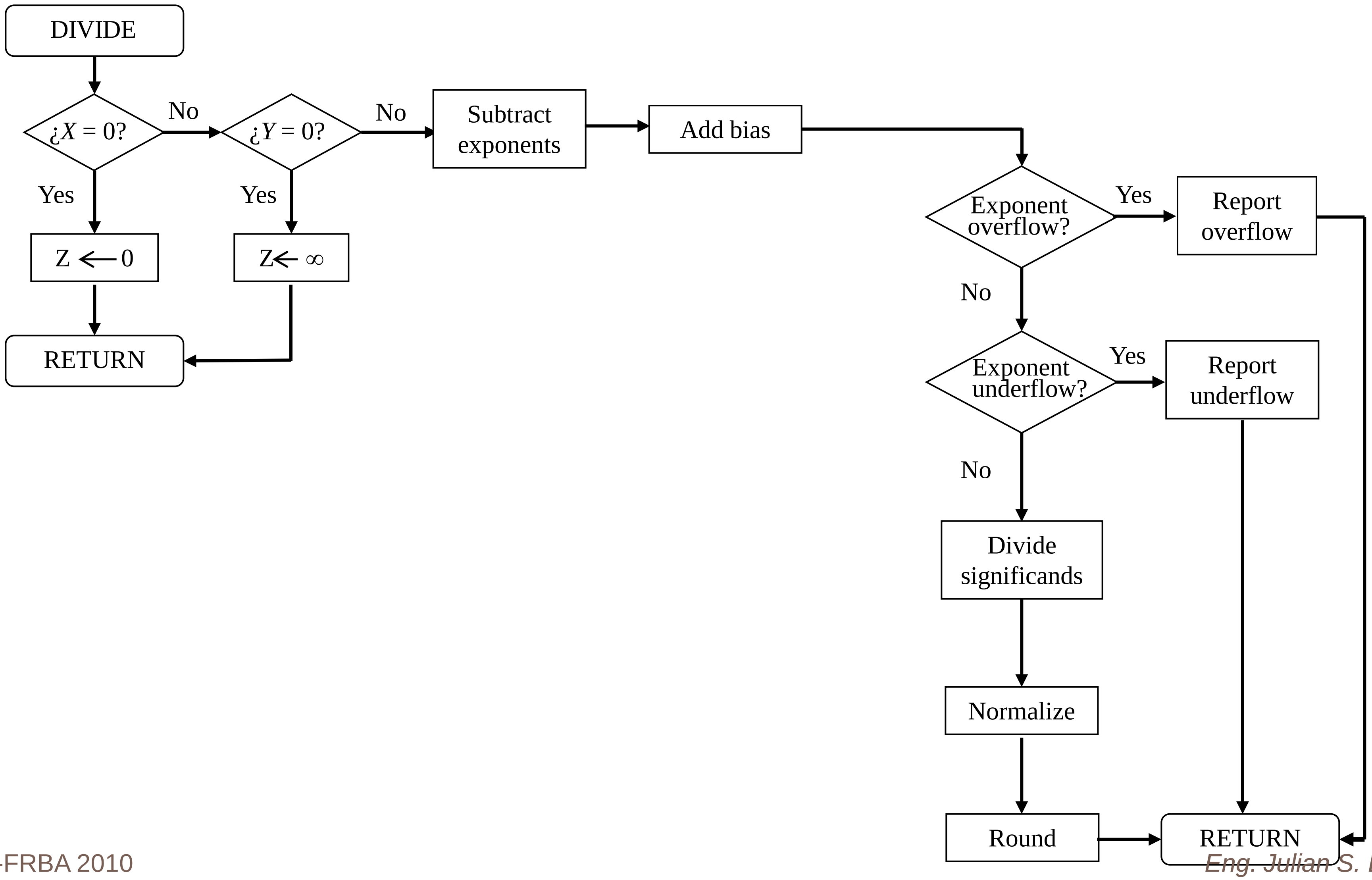
$(1 \times 2^{-23}) \times 2^{-126} \cong 2^{-149}$

1.40129846432481707092e - 45

Multiply


MULTIPLY
¿X = 0?
No
¿Y = 0?
No
Add exponents
Subtract bias
Yes
Yes
Z← 0
RETURN
Exponent overflow?
Yes
Report overflow
No
Exponent underflow?
Yes
Report underflow
No
Multiply significands
Normalize
Round
RETURN

Division


DIVIDE
¿X = 0?
No
¿Y = 0?
No
Subtract exponents
Add bias
Yes
Yes
Z ← 0
Z ← ∞
RETURN
Exponent overflow?
Yes
Report overflow
No
Exponent underflow?
Yes
Report underflow
No
Divide significands
Normalize
Round
RETURN

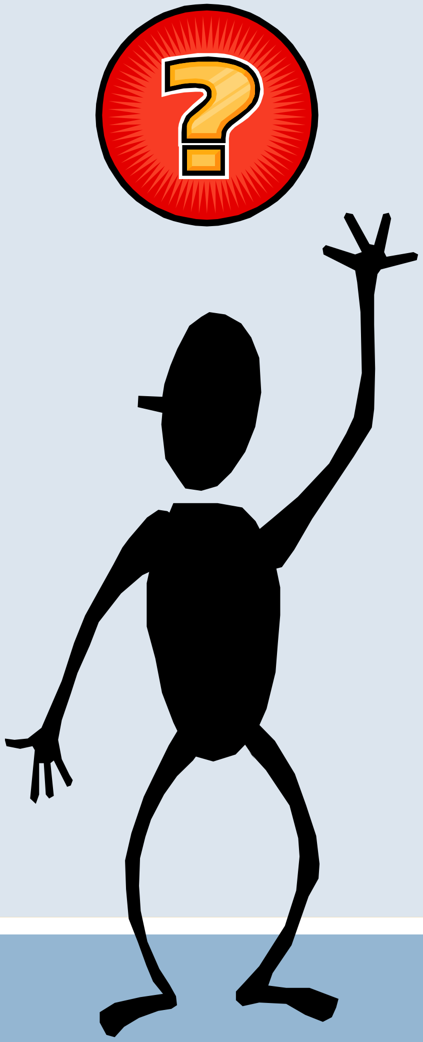
Fixed Point hardware or Floating Point Hardware ?

- There are both benefits and trade-offs to using Fix-PHw rather than FL-PHw . Many applications require low-power and cost-effective circuitry, which makes Fix-PHw a natural choice.
- Fix-PHw tends to be simpler and smaller. As a result, these units require less power and cost less to produce than floating-point circuitry.
- FL-PHw is usually larger because it demands functionality and ease of development. FL-PHw can accurately represent real-world numbers, and its large dynamic range reduces the risk of overflow, quantization errors, and the need for scaling. In contrast, the smaller dynamic range of Fix-PHw that allows for low-power, inexpensive units brings the possibility of these problems.
- Therefore, fixed-point development must minimize the negative effects of these factors, while exploiting the benefits of Fix-PHw ; cost- and size-effective units, less power and memory usage, and fast real-time processing.

Recommended bibliography

- RG Lyons, Understanding Digital Signal Processing 2nd ed. Prentice Hall. 2004.
  - Ch12: Digital Data Formats and their effects.
- SW Smith, The Scientist and Engineer's guide to DSP. California Tech. Pub. 1997.
  - Ch4: DSP software.
- VK Madisetti, DB Williams. Digital Signal Processing Handbook. CRC Press.
  - Ch3: Finite Wordlength Effects. Bruce W. Bomar
- SM Kuo, BH Lee. Real-Time Digital Signal Processing 2nd ed. John Wiley and Sons. 2006.
  - Ch 3.4 to 3.6: DSP Fundamentals and Implementations Considerations.
- WS Gan, SM Kuo. Embedded Signal Processing with the MSA. John Wiley and Sons. 2007
  - Ch 6: Real Time DSP Fundamentals and Implementations Considerations.

- NOTE: Many images used in this presentation were extracted from the recommended bibliography.



Questions?

Thank you!