7. FINITE WORD LENGTH EFFECTS IN DIGITAL FILTERS

7.1 Introduction

Digital signal processing algorithms are realized either with special purpose digital hardware or as programs for a general purpose digital computer. In both cases the numbers and coefficients are stored in finite-length registers. Therefore, coefficients and numbers must be quantized by truncation or rounding before they can be stored.

The following errors arise due to quantization of numbers.

1. Input quantization error.
2. Product quantization error.
3. Coefficient quantization error.

1. The conversion of a continuous-time input signal into digital value produces an error, which is known as input quantization error. This error arises due to the representation of the input signal by a fixed number of digits in the A/D conversion process.
2. Product quantization errors arise at the output of a multiplier. Multiplication of a b bit data with a b bit coefficient results a product having $2b$ bits. Since a b bit register is used, the multiplier output must be rounded or truncated to b bits which produces an error.
3. The filter coefficients are computed to infinite precision in theory. If they are quantized, the frequency response of the resulting filter may differ from the desired response and sometimes the filter may fail to meet the desired specifications. If the poles of the desired filter are close to the unit circle, then those of the filter with quantized coefficients may lie just outside the unit circle, leading to unstability.

The other errors arising from quantization are roundoff noise and limit cycle oscillations.

7.2 Number Representation

We can represent a number N to any desired accuracy by the finite series

$$N = \sum_{i=n_1}^{n_2} c_i r^i$$

where r is called the radix.

If $r = 10$, the representation is known as decimal representation having numbers from 0 to 9. In this representation the number

$$30.285 = \sum_{i=-3}^{1} c_i 10^i$$

$$= 3 \times 10^1 + 0 \times 10^0 + 2 \times 10^{-1} + 8 \times 10^{-2} + 5 \times 10^{-3}$$

If $r = 2$ the representation is known as binary representation with two numbers 0 and 1. For example, the binary number

$$110.010 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3}$$

has a decimal value of 6.25. To convert from decimal to binary, we divide the integer part of the number (left to the decimal point) repeatedly by 2 and arrange the remainder in reverse order. The fractional part (right to the decimal point) is repeatedly multiplied by 2, each time removing the integer part, and writing in normal order.

Example 7.1: Convert the decimal number 30.275 to binary form.

Solution

Integer part	Remainder
$30 \div 2 = 15$	0
$15 \div 2 = 7$	1
$7 \div 2 = 3$	1
$3 \div 2 = 1$	1
$1 \div 2 = 0$	1

Binary number ↑

Fractional part	Integer part
$0.275 \times 2 = 0.550$	0
$0.55 \times 2 = 1.10$	1
$0.1 \times 2 = 0.2$	0
$0.2 \times 2 = 0.4$	0
$0.4 \times 2 = 0.8$	0
$0.8 \times 2 = 1.6$	1
$0.6 \times 2 = 1.2$	1
$0.2 \times 2 = 0.4$	0

Binary number ↓

Therefore,

$$(30.275)_{10} = (11110.01000110\ldots)_2$$

7.3 Types of Number Representation

There are three common forms that are used to represent the numbers in a digital computer or any other digital hardware 1. Fixed point representation, 2. Floating point representation, 3. Block Floating point representation.

7.3.1 Fixed Point representation

In fixed-point arithmetic the position of the binary point is fixed. The bit to the right represent the fractional part of the number and those to the left represent the integer part. For example, the binary number 01.1100 has the value 1.75 in decimal.

The manner in which negative numbers are represented gives three different forms for fixed-point arithmetic.

1. Sign-magnitude form
2. One's-complement form

3. Two's-complement form

7.3.2 Sign–magnitude form

In this representation the most significant bit is set to 1 to represent the negative sign. For example, the decimal number -1.75 is represented as 11.110000 and 1.75 is represented as 01.110000. In sign-magnitude form the number 0 has two representations i.e., 00.000000 or 10.000000. With b bits only $(2^b - 1)$ numbers can be represented.

7.3.3 One's–Complement form

In one's-complement form the positive number is represented as in the sign-magnitude notation. But the negative number is obtained by complementing all the bits of the positive number. For example, the decimal number -0.875 can be represented as follows.

$$(0.875)_{10} = (0.111000)_2$$

$$(-0.875)_{10} = (1.000111)_2$$

$$\boxed{\begin{array}{l} (0.875)_{10} = (0.111000)_2 \\ \quad\quad\quad\quad \downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow \quad \text{complementing each bit} \\ \quad\quad\quad\quad 1.000111 \\ (-0.875)_{10} = (1.000111)_2 \end{array}}$$

This is same as subtracting the magnitude from $2 - 2^{-b}$, where b is the number of bits (without sign bit)

$$2 - 2^{-b} = 10.000000 - 0.000001 = 1.111111$$

Now subtract $0.875 = (0.111000)_2$

$$\begin{array}{l} 1.111111 \\ \underline{0.111000} \\ \underline{1.000111} \end{array} = (-0.875)_{10} \text{ in one's-complement form}$$

In one's-complement form the magnitude of the negative number is given by

$$1 - \sum_{i=1}^{b} c_i 2^{-i} - 2^{-b} \quad \text{... (7.1)}$$

$$\boxed{\because 1 - (2^{-4} + 2^{-5} + 2^{-6}) - 2^{-6} = 0.875}$$

In this type of representations 0 can be represented as 00.000000 and 11.111111. So with b bits $(2^b - 1)$ numbers can be represented exactly.

7.3.4. Two's-complement form

In two's-complement representation positive numbers are represented as in sign-magnitude and one's complement. The negative number is obtained by complementing all the bits of the positive number and adding one to the least significant bit.

For example,

$$(0.875)_{10} = (0.111000)_2$$

$$(-0.875)_{10} = (1.001000)_2$$

$$\boxed{\begin{array}{l} (0.875)_{10} = (0.111000)_2 \quad \downarrow \text{complementing each bit} \\ \quad\quad\quad\quad\; 1.000111 \\ \quad\quad\quad + 0.000001 \text{ Add 1 to LSB} \\ (-0.875)_{10} = \underline{1.001000} \end{array}}$$

This is same as subtracting the magnitude from 2

$$\begin{array}{rcl} 2.0 & = & 10.000000 \\ & & -0.111000 \\ \hline & & \underline{1.001000} = (-0.875)_{10} \text{ in two's-complement form} \end{array}$$

The magnitude of the negative number is given by

$$1 - \sum_{i=1}^{b} c_i 2^{-i} \quad \text{... (7.2)}$$

$$\boxed{\because\ 1 - 2^{-3} = 0.875}$$

Table 7.1 shows the comparison of the three number formats for a four bit (Including sign bit) word length. If the MSB of the binary word is one, then the number is a negative fraction and if the MSB is zero the number is a positive fraction. For this reason the MSB is called sign bit. The remaining bits are called the databits.

Table 7.1 Different Number representation for 3-bit word length (excluding sign bit)

Binary Number	Sign-magnitude	Two's-Complement	One's-Complem
0.111	7/8	7/8	7/8
0.110	6/8	6/8	6/8
0.101	5/8	5/8	5/8
0.100	4/8	4/8	4/8
0.011	3/8	3/8	3/8
0.010	2/8	2/8	2/8
0.001	1/8	1/8	1/8
0.000	0	0	0
1.000	0	– 1	– 7/8
1.001	– 1/8	– 7/8	– 6/8
1.010	– 2/8	– 6/8	– 5/8
1.011	– 3/8	– 5/8	– 4/8
1.100	– 4/8	– 4/8	– 3/8
1.101	– 5/8	– 3/8	– 2/8
1.110	– 6/8	– 2/8	– 1/8
1.111	– 7/8	– 1/8	0

7.3.5 Addition of two fixed point numbers

The addition of two fixed point numbers is simple. The two numbers added bit by bit starting from right, with carry bit being added to the ne

For example $(0.5)_{10} + (0.125)_{10}$ can be added as shown below.

Assuming the total number of bits $b + 1 = 4$ (including sign bit) we obtain

$$\begin{aligned}(0.5)_{10} &= 0.100_2\\ (0.125)_{10} &= 0.001_2\\ &\overline{0.101_2} = (0.625)_{10}\end{aligned}$$

$\llcorner$ sign bit

When two numbers of b bits are added and the sum cannot be represented by b bits an overflow is said to occur.

For example, the addition of $(0.5)_{10}$ and $(0.625)_{10}$, results $(1.125)_{10}$. If $b = 3$, then the addition of the two numbers is

$$\begin{aligned}(0.5)_{10} &= 0.100\\ (0.625)_{10} &= 0.101\\ &\overline{1.001} = (-0.125)_{10} \text{ in sign magnitude}\end{aligned}$$

$\llcorner$ sign bit

Overflow occurs in the above result because $(1.125)_{10}$ cannot be represented in the 3 bit number system. Thus in general, addition of fixed point numbers causes an overflow. The subtraction of two fixed point numbers can be performed easily by using two's complement representation.

Example 7.2: Subtract (a) 0.25 from 0.5 and (b) 0.5 from 0.25

Solution

	Decimal		Two's-complement	
(a)	0.5	=	0.100	add
	−0.25	=	1.110	
			$\overline{10.010}$	= 0.25
			neglect carry bit	

Two's complement representation of −0.25

$(0.25)_{10} = (0.010)_2$

$\downarrow\downarrow\downarrow\downarrow$ complementing each bit

1.101

+ 0.001 Add 1 to LSB

= 1.110

i.e., $(-0.25)_{10} = (1.110)_2$

Carry is generated after the addition. So the result is positive. Neglect carry bit to get the result in decimal

i.e., $(0.010)_2 = (0.25)_{10}$

(b) Decimal Two's-complement

$$\begin{array}{rcl} 0.25 & = & 0.010 \\ -0.5 & = & 1.100 \\ \hline & & \underline{1.110} \end{array} \left.\right\} \text{add}$$

Two's complement representation of -0.5

$$\begin{array}{rl} (0.5)_{10} = & (0.100)_2 \\ & \downarrow\downarrow\downarrow\downarrow \ \text{complementing each bit} \\ & 1.011 \\ + & 0.001 \ \text{Add 1 to LSB} \\ \hline = & \underline{1.100} \end{array}$$

i.e., $(-0.5)_{10} = (1.100)_2$

Carry is not generated after the addition. So the result is negative. To g the decimal output, find the two's complement of 1.110

$$\begin{array}{r} 0.001 \\ 1 \\ \hline \underline{0.010} = (0.25)_{10} \end{array}$$

i.e., $(-0.25)_{10}$

7.3.6 Multiplication in fixed point arithmetic

In multiplication of two fixed point numbers first the sign and magnitu components are separated. The magnitude of the numbers are multiplied fir then the sign of the product is determined and applied to the result. When b bit number is multiplied with another b bit number the product may con $2b$ bits. For example $(11)_2 \times (11)_2 = (1001)_2$. If the b-bits are organized in $b = b_i + b_f$, where b_i represents integer part and b_f represents the fraction the the product may contain $2\,b_i + 2\,b_f$ bits.

In fixed point arithmetic, multiplication of two fractions results a fraction

For multiplication with fractions overflow can never occur.

For example $\underset{(4\text{ bits})}{0.1001} \times \underset{(4\text{ bits})}{0.0011} = \underset{(8\text{ bits})}{0.00011011}$

7.4 Floating Point Numbers

In floating point representation a positive number is represented as $F = 2^c \cdot M$, where M, called mantissa, is a fraction such that $1/2 \leq M \leq 1$ and c, the exponent can be either positive or negative.

In decimal numbers, 4.5, 1.5, 6.5, and 0.625 have floating point representations as $2^3 \times 0.5625, 2^1 \times 0.75, 2^3 \times 0.8125, 2^0 \times 0.625$respectively.

Equivalently

$$2^3 \times 0.5625 = 2^{011} \times 0.1001$$

$$2^1 \times 0.75 = 2^{001} \times 0.1100$$

$$2^3 \times 0.8125 = 2^{011} \times 0.1101$$

$$2^0 \times 0.625 = 2^{000} \times 0.1010$$

$$\boxed{\begin{array}{c} \because\ (3)_{10} = (011)_2 \\ (0.5625)_{10} = (0.1001)_2 \\ (1)_{10} = (001)_2 \\ (0.75)_{10} = (0.1100)_2 \\ (0.8125)_{10} = (0.1101)_2 \\ (0.625)_{10} = (0.1010)_2 \end{array}}$$

Negative floating point numbers are generally represented by considering the mantissa as a fixed point number. The sign of the floating point number is obtained from the first bit of mantissa. In floating point arithmetic multiplications are carried out as follows.

Let
$$F_1 = 2^{c_1} \times M_1 \text{ and}$$
$$F_2 = 2^{c_2} \times M_2$$

Then the product $F_3 = F_1 \times F_2 = (M_1 \times M_2)\, 2^{c_1 + c_2}$

that is, the mantissas are multiplied using fixed point arithmetic and exponents are added. The product $(M_1 \times M_2)$ must be in the range of 0.25to 1.0. To correct this problem the exponent $(c_1 + c_2)$ must be altered.

Let us consider the multiplication of two numbers

$(1.5)_{10}$ and $(1.25)_{10}$

$$\boxed{\begin{array}{c} (0.75)_{10} = (0.1100)_2 \\ (0.625)_{10} = (0.1010)_2 \\ \underline{0.1100 \times 0.1010} \\ \underline{0.01111000} \end{array}}$$

$$(1.5)_{10} = 2^1 \times 0.75 = 2^{001} \times 0.1100$$

$$(1.25)_{10} = 2^1 \times 0.625 = 2^{001} \times 0.1010$$

$$\text{Now, } (1.5)_{10} \times (1.25)_{10} = (2^{001} \times 0.1100) \times (2^{001} \times 0.1010)$$

$$= 2^{010} \times 0.01111$$

Addition and subtraction of two floating point numbers are more difficult than the addition and subtraction of two fixed point numbers. To carry out addition, first adjust the exponent of the smaller number until it matches the exponent of larger number. The mantissas are then added or subtracted. Finally, the resulting representation is rescaled so that its mantissa lies in the range 0.5 to 1.

Suppose we are adding 3.0 and 0.125

$$3.0 = 2^{010} \times 0.110000$$

$$0.125 = 2^{000} \times 0.001000$$

Now we adjust the exponent of smaller number so that both exponents are equal.

$$0.125 = 2^{010} \times 0.0000100$$

Now the sum is equal to $2^{010} \times 0.110010$

7.4.1 Comparison of fixed point and floating point arithmetic

Fixed Point Arithmetic	Floating Point Arithmetic
Fast Operation	Slow Operation
Relatively economical	More expensive because of costlier hardware
Small dynamic range	Increased dynamic range
Roundoff errors occurs only for addition	Roundoff errors can occur with both addition and multiplication
Overflow occur in addition	Overflow does not arise
Used in small computers	Used in larger, general purpose computers

7.5 Block Floating Point Numbers

A compromise between fixed and floating point systems is the block-floating point arithmetic. Here, the set of signals to be handled is divided into blocks