# UNIT-IV FINITE WORDLENGTH EFFECTS

Representation of numbers - ADC Quantization noise - Coefficient Quantization error, Product Quantization error - truncation & rounding errors, Limit cycle due to product round-off error - Roundoff noise power.

## Introduction:

When digital systems are implemented either in hardware or in software, the filter coefficients are stored in binary registers. These registers can accomodate only a finite number of bits and hence the filter coefficients have to be truncated or rounded off inorder to fit into these registers.

Truncation or rounding of the data results in degradation of system performance.

Also, in digital processing systems, a continuous time input signal is sampled and quantised

inorder to get the digital signal. The process of quantisation introduces an error in the signal.

Following are errors due to finite word length effects in digital filters:

1. ADC quantisation error (due to quantization of input data by A/D convertor)
2. Coefficient quantisation error (due to quantization of filter coefficients)
3. Product quantisation error (due to rounding the product in multiplication)
4. Limit cycles due to overflow in addition.
5. Limit cycles due to zero input.

## Representation of numbers:

Rounding or truncation introduces an error whose magnitude depends on the number of bits truncated or rounded-off. Also, the characteristics of the error depends on the form of binary number representation.

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

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

where $r \rightarrow$ radix or Base

$d_i \rightarrow i^{th}$ digit of the number.

$n_2 \rightarrow$ number of integer digits.

$n_1 \rightarrow$ number of fraction digits.

When $\boxed{r=10}$ the representation is known as decimal rep. having numbers from '0 to 9'.

$d_i$

eg: $30.285 = \sum_{i=-3}^{1} d_i 10^i$

$= 3\times10^1 + 0\times10^0 + 2\times10^{-1} + 8\times10^{-2} + 5\times10^{-3}$

When $\boxed{r=2}$ the representation is known as binary representation with two numbers '0' and '1'

$d_i = 0$ (or) $1$

eg: $110.010 = \sum_{i=-3}^{2} d_i 2^i$

$= 1\times2^2 + 1\times2^1 + 0\times2^0 + 0\times2^{-1} + 1\times2^{-2} + 0\times2^{-3}$

Similary,

when $\boxed{r=8}$, $\Rightarrow$ octal representation

$\boxed{r=16}$, $\Rightarrow$ hexadecimal representation.

Pbm Convert the decimal number 30.275 to binary form.

Soln: Integer part:

$2 \mid 30$
$2 \mid 15 - 0$
$2 \mid 7 - 1$
$2 \mid 3 - 1$
$1 - 1$

$(30)_{10} \Rightarrow (11110)_2$

fractional part:

$0.275 \times 2 = 0.550 \rightarrow 0$
$0.550 \times 2 = 1.10 \rightarrow 1$
$0.10 \times 2 = 0.2 \rightarrow 0$
$0.2 \times 2 = 0.4 \rightarrow 0$
$0.4 \times 2 = 0.8 \rightarrow 0$
$0.8 \times 2 = 1.6 \rightarrow 1$
$0.6 \times 2 = 1.2 \rightarrow 1$
$0.2 \times 2 = 0.4 \rightarrow 0$

$(0.275)_{10} \Rightarrow (01000110\ldots)_2$

Ans: $(30.275)_{10} \Rightarrow (11110.01000110.)_2$

## Types of Number Representation:

1. Fixed point Representation
2. Floating " "
3. Block floating " "

## 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 eg: 01.1100 has the value 1.75 in decimal.

there is only one unique way of representing binary fraction positive number,

$d_0$

i.e, $N_p = 0 \cdot d_1 d_2 d_3 \ldots d_{n_1}$

$= (0 \times 2^0) + (d_1 \times 2^{-1}) + (d_2 \times 2^{-2}) + \ldots + (d_{n_1} \times 2^{-n_1})$

where $d_0 = 0 \Rightarrow$ Sign bit (MSB Bit) (= 0 for positive number)

there are 3 different formats for representing negative binary numbers.

1. Sign-magnitude format
2. One's Complement "
3. Two's Complement "

$$N_p = (0 \times 2^0) + \sum_{i=1}^{n} d_i 2^{-i}$$

## Sign- Magnitude format:

In this format, the Most Significant Bit (MSB) bit is set to '1' to represent the negative sign.

For eg: $+1.75 \longrightarrow (01.1100)_2$

$-1.75 \Rightarrow (11.1100)_2$

In this format, the number '0' has two representations i.e., $00.\overset{(+0)}{0}0000$ (or) $10.\overset{(-0)}{0}0000$.

with 'b' bits, only $(2^b - 1)$ numbers can be representated. $\boxed{N_n = (1 \times 2^0) + \sum_{i=1}^{n} d_i 2^{-i}}$

**Prob eg:** Express the fraction +7/8, -7/8, +6/8, -6/8 in sign-magnitude format.

**Ans:** $(7/8)_{10} = (0.875)_{10}$

$(+0.875)_{10} \Rightarrow$ $0.875 \times 2 = 1.750 \longrightarrow 1$

$0.750 \times 2 = 1.500 \longrightarrow 1$

$0.5 \times 2 = 1.0 \longrightarrow 1$

$\Rightarrow (0.111)_2$ (0 ↓ sign bit)

$(-0.875)_{10} \Longrightarrow (1.111)_2$ (1 ↓ sign bit)

sign bit = { 0 for +ve no. ; 1 for -ve no.

$(6/8) \Longrightarrow (0.75)_{10}$

$(+6/8)_{10} \Rightarrow (0.75)_{10}$ $= 0.75 \times 2 = 1.500 \longrightarrow 1$

$0.5 \times 2 = 1.0 \longrightarrow 1$

$\Rightarrow (0.110)_2$

$(-6/8)_{10} \Rightarrow (1.110)_2$

## One's Complement format:

In this format, positive number is represented as in the sign-magnitude notation. But the negative number is obtained by complementing all the bits of the postive number.

$$N_n = (1 \times 2^0) + \sum_{i=1}^{n} (1 - d_i) 2^{-i}$$

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

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

Here the number '0' can be represented as $0.\overset{(+0)}{0}00000$ and $1.\overset{(-0)}{1}11111$.

with 'b' bits, $(2^b - 1)$ numbers can be represented exactly.

Eg. Pbm: Express the fraction +7/8, -7/8, +6/8, -6/8 in one's magnitude format.

Soln: $(+7/8)_{10} = (0.875)_{10} = (0.111)_2$

$(-7/8)_{10} = (1.000)_2$ ← complement of all bits

$(+6/8)_{10} = (0.110)_2$

$(-6/8)_{10} = (1.001)_2$

## Two's Complement format:

In this format, positive numbers are represented as in sign-magnitude and one's complement form. The negative number is obtained by complementing all the bits of the positive number and adding one to LSB.

$$N_n = (1 \times 2^0) + \sum_{i=1}^{n} (1 - d_i) 2^{-i} + (1 \times 2^{-n})$$

for eg: $(+0.875)_{10} \Longrightarrow (0.111000)_2$

$1.000111$ ← ones complement

$\underline{1}$ ← add 1 to LSB

$(-0.875)_{10} \Leftarrow \underline{(1.001000)_2}$

Pbm eg: Express the fraction $+7/8$, $-7/8$, $+6/8$, $-6/8$ in 2's complement format with 4 bit representation including sign bit.

Soln: $(+7/8)_{10} = (0.875)_{10} = (0.111)_2$

$1.000$

$\underline{1}$

$\underline{1.001}$

$(-7/8)_{10} = (1.001)_2$

$(+6/8)_{10} = (0.75)_{10} = (0.110)_2$

$1.001$

$\underline{1}$

$\underline{1.010}$

$(-6/8) = (1.010)_2$

## Comparison of 3 formats for 4-bit word length (including sign-bit)

| Decimal Equivalent | Binary no. in fixed point representation | | |
|---|---|---|---|
| | Sign-magnitude | 1's complement | 2's complement |
| 0 | 0.000 | 0.000 | 0.000 |
| 1/8 | 0.001 | 0.001 | 0.001 |
| 2/8 | 0.010 | 0.010 | 0.010 |
| 3/8 | 0.011 | 0.011 | 0.011 |
| 4/8 | 0.100 | 0.100 | 0.100 |
| 5/8 | 0.101 | 0.101 | 0.101 |
| 6/8 | 0.110 | 0.110 | 0.110 |
| 7/8 | 0.111 | 0.111 | 0.111 |
| -0 | 1.000 | 1.111 | |
| -1/8 | 1.001 | 1.110 | 1.111 |
| -2/8 | 1.010 | 1.101 | 1.110 |
| -3/8 | 1.011 | 1.100 | 1.101 |
| -4/8 | 1.100 | 1.011 | 1.100 |
| -5/8 | 1.101 | 1.010 | 1.011 |
| -6/8 | 1.110 | 1.001 | 1.010 |
| -7/8 | 1.111 | 1.000 | 1.001 |
| -8/8 | | | 1.000 |

b=4

. total: $2^4 - 1$ = 15 rep.

$= 2^4 - 1$ = 15 rep

$= 2^4$ representations

## Addition of two fixed point numbers:

two numbers are added bit by bit starting from right, with carry bit being added to next bit.

eg: $(0.5)_{10} + (0.125)_{10} = ?$ (Assume 4 bits including sign bit)

$(0.5)_{10} = 0.100_2$

$(0.125)_{10} = 0.001_2$

$0.101_2 \Rightarrow (0.625)_{10}$

↓ sign bit

Binary to decimal conversion:

$0.101$

$1 \times 2^{-3} = 0.125$

$0 \times 2^{-2} = 0.000$

$1 \times 2^{-1} = 0.500$

$0 \times 2^{0} = 0.000$

$0.625$

↓ sign bit

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

for eg: $(0.5)_{10} + (0.625)_{10} = ?$

$(0.5)_{10} = 0.100_2$

$(0.625)_{10} = 0.101_2$

$1.001_2 \Rightarrow (-0.125)_{10}$ in sign magnitude format

↓ Sign bit

Binary in decimal:

$1.001_2 \Rightarrow 1 \times 2^{-3} = 0.125$

$0 \times 2^{-2} = 0.000$

$0 \times 2^{-1} = 0.000$

$1 \times 2^{0} = 1.000$

$1.125_{10}$

general.

overflow occurs. it cannot be represented in 'b' bit number system.

## Subtraction of 2 fixed point numbers:

Subtraction is done by addition both in one's and 2's complement format.

### procedure for 1's complement addition:

Steps:
1. find 1's complement of negative number
2. Add positive number with 1's complement of negative number.
3. If carry is generated, it is added to the LSB of addition result, If no carry, Answer will be in 1's complement form.

Ex: $(0.625)_{10} - (0.375)_{10} = ?$
(or)
$(0.625)_{10} + (-0.375)_{10} = ?$

$(0.625)_{10} \Rightarrow 0.101_2$

$(0.375)_{10} \Rightarrow 0.011_2$

1's complement of $(0.375)_{10} \Rightarrow 1.100_2$

$(0.625)_{10} \Rightarrow 0.101_2$ (+)

carry $\leftarrow$ [1] $0.001$ ....→ 1

Ans: $\Rightarrow$ $0.010_2 \Rightarrow (0.250)_{10}$

Note: Subtraction of smaller number from larger number results in 'carry = 1', ∴ add carry to LSB to get original result.

Ex: $(0.375)_{10} - (0.625)_{10} = ?$

(or)

$(0.375)_{10} + (-0.625)_{10} = ?$

$0.011_2$ ⇓ $0.101_2$ ⇓ complement $1.010_2$

$(0.375)_{10} = 0.011_2$

$(-0.625)_{10} = 1.010_2$

$1.1010_2$ → result in 1's complement form.

no carry (or) carry = 0.

$1.010_2 \Rightarrow (-0.250)_{10}$ ← original result.

Note: Subtraction of larger number from smaller number results no carry. Original result is obtained by complementing the addition result except sign bit.

Procedure for 2's complement addition:

Steps:
1. find 2's complement of negative number.
2. Add the positive number with 2's complement of negative number.
3. If carry is generated, omit the carry to get original result. If no carry, answer will be in 2's complement form. Hence find 2's complement of addition result to get original result except sign bit.

Ex: $(0.625)_{10} - (0.375)_{10} = ?$

(or)

$(0.625)_{10} + (-0.375)_{10} = ?$

$0.101_2$

$0.011$

$1.100 \Rightarrow$ 1's complement

$1 \Rightarrow$ add 1 to LSB

$1.101_2$

$1.101_2$

$0.010$

carry omitted

Answer $\Rightarrow$ ~~0.101~~ $(0.010)_2 \Rightarrow (0.250)_{10}$

Subtraction of smaller number from larger number results in "carry".

Ex: $(0.375)_{10} - (0.625)_{10} = ?$

(or)

$(0.375)_{10} + (-0.625)_{10} = ?$

$0.011$

$1.011_2$

$1.110$

no carry

$0.101$

$1.010$

$1$

$1.011_2$

$001 \Rightarrow$ 1's complement

$1 \Rightarrow$ add 1 to LSB

$1.010 \Rightarrow$ original result : $(1.010)_2 \Rightarrow (-0.250)_{10}$

Subtraction of larger number from smaller number results in "no carry"

## Multiplication in fixed point arithmetic:

In multiplication of 2 fixed point numbers, ~~are~~ first sign and magnitude components are separated. The magnitude of the numbers are multiplied first, then 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 contain '2b' bits.

b=2

eg: $(11)_2 \times (11)_2 = (1001)_2$

```
  1 1
  1 1
-----
  1 1
 1 1
-------
1 0 0 1   ← 2b bits
```

Note: In fixed point arithmetic, multiplication of 2 fractions result in a fraction.

For multiplication with fractions overflow can never occur.

eg: $0.1001 \times 0.0011 = 0.00011011$

4 bits  4 bits  8 bits.

```
          1 0 0 1
          0 0 1 1
-------------------
          1 0 0 1
        1 0 0 1
      0 0 0 0
    0 0 0 0
-------------------
0 0 0 1 1 0 1 1
-------------------
```

## Floating Point Representation:

In floating point representation, a positive number is represented as

$$F = M \times 2^e$$

where M → mantissa, is a fraction $0.5 \le M \le 1.0$

e → exponent can be either +ve or -ve.

eg: $(4.5)_{10} \Rightarrow$ 2 | 4 ; 2 | 2 − 0 ; 1 − 0 ↑

$0.5 \times 2 = 1.0$ → 1 ; 0 0 ↓

$\Rightarrow (100.10\ldots)_2$

$\Rightarrow 0.1001 \times 2^3 = \boxed{0.1001 \times 2^{011}}$

$(0.625)_{10} \Rightarrow 0.1010 \times 2^0 = \boxed{0.1010 \times 2^{000}}$

$(6.5)_{10} \Rightarrow$ 2 | 6 ; 2 | 3 − 0 ↑ ; 1 − 1

$0.5 \times 2 = 1.0$ → 1 ; 0 0 ↓

$= (110.10\ldots)_2$

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

$(1.5)_{10} = (1.100\ldots)_2 = 0.1100 \times 2^1$

$= \boxed{0.1100 \times 2^{001}}$

in decimal

425.3

$= 0.4253 \times 10^3$

$= 42530 \times 10^{-2}$

Negative floating point numbers are generalled represented by considering mantissa as a fixed point number. The sign of the floating point number is obtained from the first bit of mantissa.

Multiplication of 2 floating point numbers:

$F_1 = 2^{e_1} \times M_1$ , $F_2 = 2^{e_2} \times M_2$

$F_3 = F_1 \times F_2 = (M_1 \times M_2)\, 2^{(e_1+e_2)}$

must be: 0.25 to 1.0

eg: $(1.5)_{10} \times (1.25)_{10} = ?$

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

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

$0.1100 \times 0.1010 = 0.01111000$ (8 bits)

$2^{001} \times 2^{001} \Rightarrow 2^{010} \Leftarrow (001+001)$

Ans: $0.01111000 \times 2^{010}$

```
  1010
  1100
 ------
  0000
 0000
1010
1010
-------
1111000
-------
```

## Addition and Subtraction of 2 floating point nos:

⇒ more difficult than that of fixed point nos. To carry out addition, first adjust the exponent of the smaller number until it matches the exponent of larger number. The mantissa are then added or subtracted. Finally, the resulting representation is rescaled so that its mantissa lies in the range 0.5 to 1.0

eg: $(3.0)_{10} + (0.125)_{10} = ?$

$(11.00\ldots)_2$

$0.110000 \times 2^{010}$

$0.001000 \times 2^{000}$

$= 0.0000100 \times 2^{010}$

$0.1100000 \times 2^{010}$

Answer: $0.1100100 \times 2^{010}$

## Comparison of fixed and floating point arithmetic:

| fixed | floating |
|---|---|
| 1. Fast operation | 1. Slow operation |
| 2. Small dynamic range | 2. Large dynamic range. |
| 3. Round off errors occur only for addition | 3. occurs for both addition & multiplication |
| 4. overflow occurs in addition | 4. overflow does not arise. |
| 5. hardware implementation is cheaper | 5. costlier |
| 6. preferred for real time applications | 6. non-real time applications |
| 7. used in small computers | 7. used in larger, general purpose computers. |

## 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. Each block have the same value for exponent.

The arithmetic operations within the block uses fixed point arithmetic and only one exponent per block is stored, thus saving memory. This type of representation is most suitable in certain FFT flowgraphs and in digital audio applications.