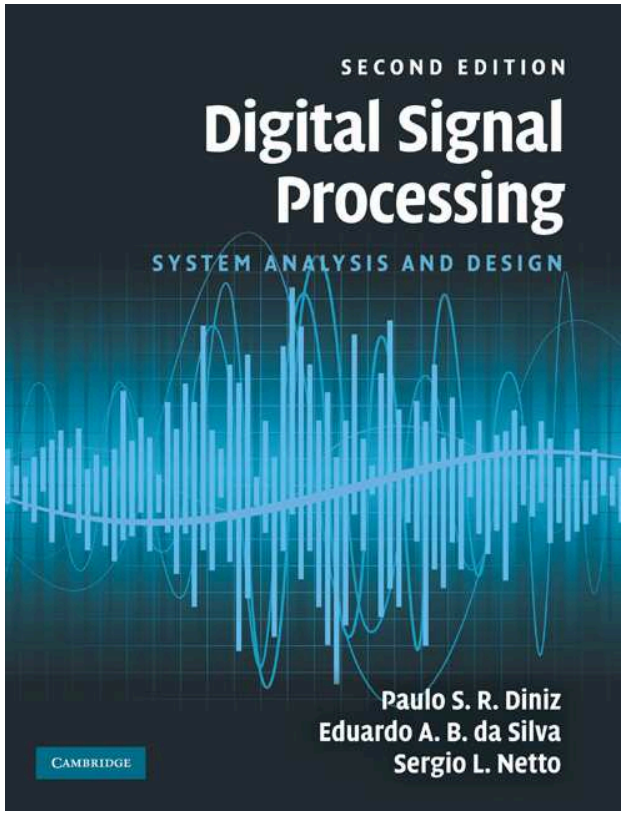
Finite-Precision Digital Signal Processing



Paulo S. R. Diniz

Eduardo A. B. da Silva

Sergio L. Netto

diniz,eduardo,sergioln@lps.ufrj.br

September 2010

Contents

- Binary number representation
  - Fixed-point representation
  - Signed power-of-two representation
  - Floating-point representation
- Basic elements
  - Properties of the two's-complement representation
  - Serial adder
  - Serial multiplier
  - Parallel adder
  - Parallel multiplier

Contents (cont.)

- Distributed arithmetic implementation
- Product quantization
- Signal scaling
- Coefficient quantization
  - Deterministic sensitivity criterion
  - Statistical forecast of the wordlength

Contents (cont.)

- Limit cycles
  - Granular limit cycles
  - Overflow limit cycles
  - Elimination of zero-input limit cycles
  - Elimination of constant-input limit cycles
  - Forced-response stability of digital filters with nonlinearities due to overflow
- Do-it-yourself: Finite-precision digital signal processing

Introduction

- This chapter starts by addressing some implementation methods for digital filtering algorithms and structures.
- The implementation of any building block of digital signal processing can be performed using a software routine on a simple personal computer.
- In this case, the designer's main concern becomes the description of the desired filter as an efficient algorithm that can be easily converted into a piece of software. In such cases, the hardware concerns tend to be noncritical, except for some details such as memory size, processing speed, and data input/output.

Introduction

- Another implementation strategy is based on specific hardware, especially suitable for the application at hand.
- In such cases, the system architecture must be designed within the speed constraints at a minimal cost.
- This form of implementation is mainly justified in applications that require high processing speed or in large-scale production.

Introduction

- The four main forms of appropriate hardware for implementing a given system are:
  - The development of a specific architecture using basic commercial electronic components and integrated circuits.
  - The use of programmable logic devices (PLD), such as field-programmable gate arrays (FPGA), which represent an intermediate integrated stage.
  - The design of a dedicated integrated circuit for the application at hand using computer-automated tools for a very large scale integration (VLSI) design. The basic goal in designing application-specific integrated circuits (ASIC) is to obtain a final design satisfying specifications with respect to chip area, power consumption, processing speed, testability, and overall production cost.
  - The use of a commercially available general-purpose digital signal processor (DSP). There are several commercial DSPs available today, which include features such as fixed- or floating-point operation, several ranges of clock speed and price, internal memory, and very fast multipliers.

Introduction

- The state-of-the-art implementation of digital signal processing systems falls beyond the scope of this book. Instead we address some fundamental concepts related to hardware implementation.
- This chapter starts by discussing the most widely used binary number representations in digital signal processing.
- Then, we introduce the basic elements necessary for the implementation of systems for digital signal processing, in particular the digital filters extensively covered by this book.
- Distributed arithmetic is presented as a design alternative for digital filters that eliminates the necessity of multiplier elements. These very basic implementation concepts illustrate issues related to hardware implementation of digital signal processing systems.

Introduction

- It is certain that in practice a digital signal processing system is implemented by software on a digital computer, either using a general-purpose digital signal processor, or using dedicated hardware for the given application.
- In either case, quantization errors are inherent due to the finite-precision arithmetic. These errors are of the following types:
  - Errors due to the quantization of the input signals into a set of discrete levels, such as the ones introduced by the analog-to-digital converter.
  - Errors in the frequency response of filters, or in transform coefficients, due to the finite-wordlength representation of multiplier constants.
  - Errors made when internal data, like outputs of multipliers, are quantized before or after subsequent additions.

Introduction

- All these error forms depend on the type of arithmetic utilized in the implementation.
- If a digital signal processing routine is implemented on a general-purpose computer, since floating-point arithmetic is in general available, this type of arithmetic becomes the most natural choice.
- On the other hand, if the building block is implemented on a special-purpose hardware or a fixed-point digital signal processor, fixed-point arithmetic may be the best choice, because it is less costly in terms of hardware and simpler to design. A fixed-point implementation usually implies a lot of savings in terms of chip area as well.
- For a given application, the quantization effects are key factors to be considered when assessing the performance of a digital signal processing algorithm.

Introduction

- In this chapter, the various quantization effects are introduced along with the most widely used formulations for their analyses.
- In particular, the product and coefficient quantization effects are discussed in some detail along with the techniques to scale the internal signals in order to avoid frequent overflows.
- The chapter also discusses strategies to eliminate zero-input and constant-input granular limit cycles and to avoid sustained overflow oscillations.
- The Do-it-yourself section illustrates some finite precision issues in a practical example.

Binary number representation

- In this section, we consider three numerical representations using binary codes:
  - Fixed-point representations
  - Signed power-of-two representation
  - Floating-point representation
- In the following subsections we define each of these representations, emphasizing their advantages and numerical properties.

Fixed-point representations

- In most of the cases when digital signal processing systems are implemented using fixed-point arithmetic, the numbers are represented in one of the following forms: sign-magnitude, one's-complement, and two's-complement formats.
- These representations are described in this subsection, where we implicitly assume that all numbers have been previously scaled to the interval $x \in (-1, 1)$.
- In order to clarify the definitions of the three types of number representation given below, we associate, for every positive number x such that $x < 2$, a function that returns its representation in base 2, $\mathcal{B}(x)$, which is defined by the equations below:

$$\mathcal{B}(x) = x_0.x_1x_2\cdots x_n \tag{1}$$

and

$$x = \mathcal{B}^{-1}(x_0.x_1x_2\cdots x_n) = x_0 + x_1 2^{-1} + x_2 2^{-2} + \cdots + x_n 2^{-n} \tag{2}$$

Fixed-point representations - Sign-magnitude truncation

- The sign-magnitude representation of a given number consists of a sign bit followed by a binary number representing its magnitude. That is

$$[x]_M = s_x.x_1x_2x_3\cdots x_n \tag{3}$$

where s_x is the sign bit, and $x_1x_2x_3\cdots x_n$ represents the magnitude of the number in base 2, that is, in binary format.

- Here, we use $s_x = 0$ for positive numbers, and $s_x = 1$ for negative numbers. This means that the number x is given by

$$x = \begin{cases} \mathcal{B}^{-1}(0.x_1x_2\cdots x_n) = x_1 2^{-1} + x_2 2^{-2} + \cdots + x_n 2^{-n}, & \text{for } s_x = 0 \\ -\mathcal{B}^{-1}(0.x_1x_2\cdots x_n) = -(x_1 2^{-1} + x_2 2^{-2} + \cdots + x_n 2^{-n}), & \text{for } s_x = 1 \end{cases} \tag{4}$$

Fixed-point representations - One's-complement representation

- The one's-complement representation of a number is given by

$$[x]_{1c} = \begin{cases} \mathcal{B}(x), & \text{if } x \geq 0 \\ \mathcal{B}(2 - 2^{-n} - |x|), & \text{if } x < 0 \end{cases} \tag{5}$$

  where $\mathcal{B}$ is defined by equations (1) and (2).

- Notice that in the case of positive numbers, the one's-complement and the sign-magnitude representations are identical.

- However, for negative numbers, the one's complement is generated by changing all the 0s to 1s and all the 1s to 0s in the sign-magnitude representation of its absolute value.

- As before $s_x = 0$ for positive numbers and $s_x = 1$ for negative numbers.

Fixed-point representations - Two's-complement representation

- The two's-complement representation of a number is given by

$$[x]_{2c} = \begin{cases} \mathcal{B}(x), & \text{if } x \geq 0 \\ \mathcal{B}(2 - |x|), & \text{if } x < 0 \end{cases} \tag{6}$$

  where $\mathcal{B}$ is defined by equations (1) and (2).

- Again, for positive numbers, the two's-complement representation is identical to the sign-magnitude representation.

- The representation of a negative number in two's complement can be obtained by adding 1 to the least significant bit of the number represented in one's complement.

- Then, as in the one's-complement case, $s_x = 0$ for positive numbers and $s_x = 1$ for negative numbers.

Fixed-point representations - Two's-complement representation

- If the two's-complement representation of a number x is given by

$$[x]_{2c} = s_x.x_1x_2\cdots x_n \tag{7}$$

then, from equation (6), we have that

  - For $s_x = 0$, then

$$x = \mathcal{B}^{-1}([x]_{2c}) = \mathcal{B}^{-1}(0.x_1x_2\cdots x_n) = x_12^{-1} + x_22^{-2} + \cdots + x_n2^{-n} \tag{8}$$

  - For $s_x = 1$, then

$$2 - |x| = 2 + x = \mathcal{B}^{-1}([x]_{2c}) = \mathcal{B}^{-1}(s_x.x_1x_2\cdots x_n) \tag{9}$$

and then

$$\begin{aligned} x &= -2 + \mathcal{B}^{-1}(1.x_1x_2\cdots x_n) \\ &= -2 + 1 + x_12^{-1} + x_22^{-2} + \cdots + x_n2^{-n} \\ &= -1 + x_12^{-1} + x_22^{-2} + \cdots + x_n2^{-n} \end{aligned} \tag{10}$$

Fixed-point representations - Two's-complement representation

- From equations (8) and (10), we have that x can be generally expressed as

$$x = -s_x + x_1 2^{-1} + x_2 2^{-2} + \cdots + x_n 2^{-n} \tag{11}$$

- This short notation is very useful for introducing the multiplication operation of binary numbers represented in two's-complement format, as given in Section 11.3.
- The one's-complement and two's-complement representations are more efficient for addition implementations, whereas the sign-magnitude representation is more efficient for the implementation of multiplications. Overall, the most widely used binary code is the two's-complement representation.

Example 11.1

- Represent the number -0.1875 using 8 bits (including the sign bit) in one's-complement and two's-complement representations.

Example 11.1 - Solution

- We first obtain the standard binary representation for the absolute value of the number, using the following procedure: Multiply 0.1875 by 2 and associate a bit 0, if the product is less than 1.0, or a bit 1, if the product is greater than or equal to 1.0.
- In this last case, subtract 1.0 from the product, to bring the partial result back to below 1.0.

Example 11.1 - Solution

- Repeat this process for each required bit, yielding, in this case, the computations:

Multiplication by 2	Bit
$0.1875 \times 2 = 0.375$	0
$0.375 \times 2 = 0.75$	0
$0.75 \times 2 = 1.5$	1
$0.5 \times 2 = 1.0$	1
$0.0 \times 2 = 0$	0
$0.0 \times 2 = 0$	0
$0.0 \times 2 = 0$	0

(12)

- Hence, the binary representation of 0.1875 is 0.0011000. By changing all bits 0 by bits 1 and vice versa, we obtain the one's-complement representation 1.1100111 of -0.1875. The two's-complement representation is computed by adding 1 to the least significant bit of the one's-complement representation, yielding 1.1101000.

Signed power-of-two representation

- An alternative representation of a number consisting of weighted sums and subtractions is referred to as the signed-digit (SD) representation. The main particular case of SD is the signed power-of-two (SPT) representation, also known as radix-two signed-digit code.
- The SPT representation allows a multiplication to be performed as a very simple sequence of shifts, additions, and subtractions, which is very attractive from the hardware implementation point of view.
- A given number x is represented in SPT as

$$x = x_0 + x_1 2^{-1} + x_2 2^{-2} + \cdots + x_n 2^{-n} \tag{13}$$

where $x_i \in \{-1, 0, 1\} = \{\bar{1}, 0, 1\}$.

Signed power-of-two representation

- Note that no sign digit is required in the SPT format. For instance, the number -0.1875 can be represented in SPT format as

$$x_0x_1x_2x_3x_4 = \begin{cases} 000\bar{1}\bar{1} \\ 00\bar{1}01 \\ 00\bar{1}1\bar{1} \\ 0\bar{1}101 \\ 0\bar{1}11\bar{1} \end{cases} \tag{14}$$

- Since the SPT representation of a number is not unique, representations with the maximum number of 0 digits, which may also not be unique, are desirable from the computational point of view.

- If, in addition, we avoid two consecutive nonzero digits, the so-called canonic signed-digit (CSD) representation arises, which can be shown to be unique.

Signed power-of-two representation

- For instance, the second SPT representation in equation (14) is the corresponding CSD code for -0.1875.
- Given an $(n+1)$-bit two's-complement representation $[x]_{2c} = s_x.x_1x_2\cdots x_n$ of a number, it can be transformed into a CSD representation $[x']_{CSD} = x'_0x'_1\cdots x'_n$ through the following algorithm:

  (i) Initialization: Set the auxiliary variables $x_{-1} = s_x$ and $x_{n+1} = \delta_{n+1} = 0$.

  (ii) For $i = n, (n-1), \ldots, 0$ determine, in sequence,

$$\theta_i = x_i \oplus x_{i+1} \tag{15}$$

$$\delta_i = \overline{\delta_{i+1}} \times \theta_i \tag{16}$$

$$x'_i = (1 - 2x_{i-1})\delta_i \tag{17}$$

  where $\overline{[\cdot]}$, $\times$, and $\oplus$ represent the binary complement, AND, and exclusive-OR operations, respectively.

Signed power-of-two representation

- The main advantage of the SPT representation over the two's-complement is that the inclusion of signed-digits leads to a reduction in the number of nonzero digits, simplifying subsequent arithmetic operations.
- Using the CSD representation, for instance, the number of nonzero digits tends to $\frac{3n+4}{9}$ as n increases.
- The hardware reduction property of the CSD representation has been exploited in several specialized filter design approaches.

Floating-point representation

- Using floating-point representation, a number is represented as

$$x = x_m 2^c \tag{18}$$

  where x_m is the mantissa and c is the number exponent, with $\frac{1}{2} \leq |x_m| < 1$.

- For example, the number -0.001875 can be represented in floating point by 1.1101000×2^{-2} where the mantissa has a two's-complement representation.

- When compared to fixed-point representations, the main advantage of the floating-point representation is its large dynamic range. Its main disadvantage is that its implementation is more complex.

- For example, in floating-point arithmetic, the mantissa must be quantized after both multiplications and additions, whereas in fixed-point arithmetic quantization this is required only after multiplications.

- In this text, we deal with both fixed- floating-point arithmetics, with more focus on the fixed-point case, which is more prone to errors and requires more attention.

Basic elements

- In this section, we study four basic elements for digital signal processing, namely: serial adder, parallel adder, serial multiplier, and parallel multiplier.
- Before that, we introduce some properties of the two's-complement representation.

Properties of the two's-complement representation

- Since the two's-complement representation is vital to the understanding of the arithmetic operations described in the following sections, here we supplement the introduction given in Section 11.2, giving some other properties of the two's-complement representation.
- We start by repeating equation (11), which states that if the two's-complement representation of x is given by

$$[x]_{2c} = s_x.x_1x_2\cdots x_n \tag{19}$$

then, the value of x is determined by

$$x = -s_x + x_1 2^{-1} + x_2 2^{-2} + \cdots + x_n 2^{-n} \tag{20}$$

Properties of the two's-complement representation

- One of the advantages of the two's-complement representation is that, if A and B are represented in two's complement, then $C = A - B$, represented in two's complement, can be computed by adding A to the two's complement of B. Also, given x as in equation (20), we have that

$$\frac{x}{2} = -s_x 2^{-1} + x_1 2^{-2} + x_2 2^{-3} + \cdots + x_n 2^{-n-1} \tag{21}$$

and since $-s_x 2^{-1} = -s_x + s_x 2^{-1}$, then equation (21) can be rewritten as

$$\frac{x}{2} = -s_x + s_x 2^{-1} + x_1 2^{-2} + x_2 2^{-3} + \cdots + x_n 2^{-n-1} \tag{22}$$

Properties of the two's-complement representation

- That is, the two's-complement representation of $\frac{x}{2}$ is given by

$$\left[\frac{x}{2}\right]_{2c} = s_x.s_x x_1 x_2 \cdots x_n \tag{23}$$

- This property implies that dividing a number represented in two's complement by 2 is equivalent to shifting all of its bits one position to the right, with the extra care of repeating the sign bit.

- This property is very important in the development of the multiplication algorithm, which involves multiplications by 2^{-j}.

Serial adder

- A very economic implementation of digital filters is achieved through the use of so-called serial arithmetic. Such an approach performs a given operation by processing the bits representing a given binary number one by one serially.
- The overall result tends to be very efficient in terms of required hardware, power consumption, modularity, and ease of interconnection between serial-bit cells.
- The main drawback is clearly the associated processing speed, which tends to be very slow, when compared with other approaches.
- An important basic element for the implementation of all signal processing systems is the full adder whose symbol and respective logic circuit are shown in Figure 1.
- Such a system presents two output ports, one equal to the sum of the two input bits A and B, and the other, usually referred to as the carry bit, corresponding to the possible extra bit generated by the addition operation.

Serial adder

- A third input port C_i is used to allow the carry bit from a previous addition to be taken into account in the current addition.
- A simple implementation of a serial adder for the two's-complement arithmetic, based on the full adder, is illustrated in Figure 2a.
- In such a system, the two input words A and B must be serially fed to the adder starting with their least-significant bit.
- A D-type flip-flop (labeled D) is used to store the carry bit from a given 1-bit addition, saving it to be used in the addition corresponding to the next significant bit.
- A reset signal is used to clear this D-type flip-flop before starting the addition of two other numbers, forcing the carry input to be zero at the beginning of the summation.

Serial adder

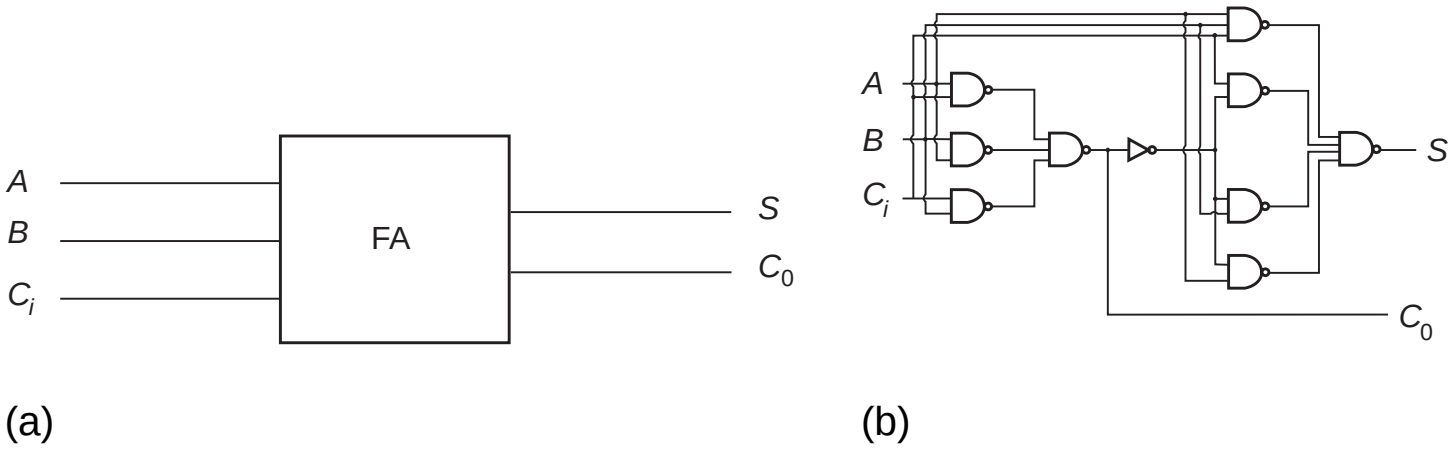
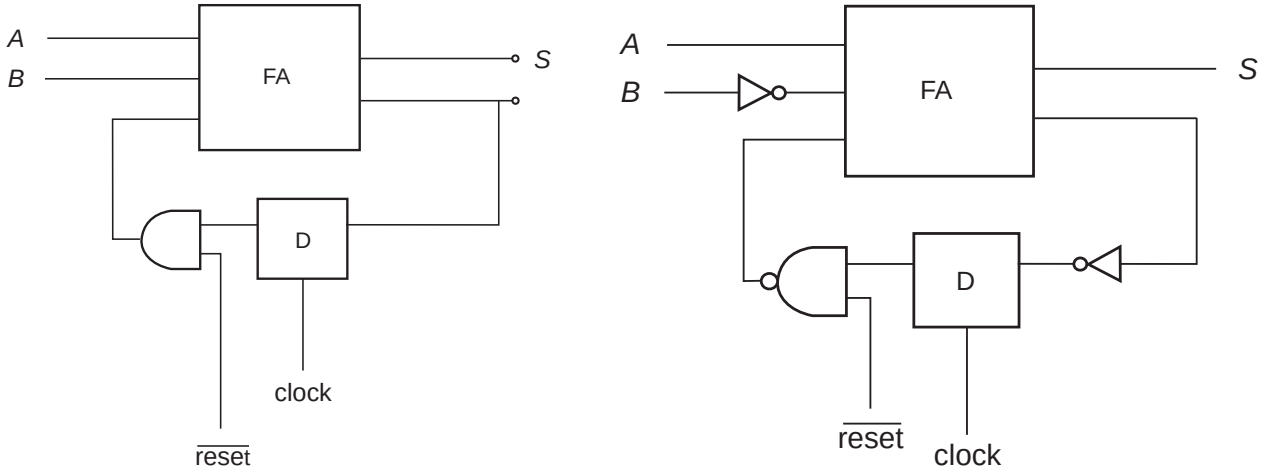


Figure 1: Full adder: (a) symbol; (b) logic circuit.

Serial adder



(a) (b)

Figure 2: Serial implementations using the full adder as a basic block: (a) adder; (b) subtractor.

Serial adder

- Figure 2b depicts the serial subtractor for the two's-complement arithmetic.
- This structure is based on the fact that $A - B$ can be computed by adding A to the two's complement of B, which can be determined by inverting B and summing 1 in the least-significant bit.
- The representation of B in two's complement is commonly done with an inverter at the input of B and with a NAND gate, substituting the AND gate, at the carry input.
- Then, at the beginning of the summation, the signal `reset` is asserted, and the carry input becomes 1.
- An extra inverter must be placed at the carry output because, with the use of the NAND gate, the carry output fed back to the D-type flip-flop must be inverted.

Serial multiplier

- The most complex basic element for digital signal processing is the multiplier.
- In general, the product of two numbers is determined as a sum of partial products, as performed in the usual multiplication algorithm.
- Naturally, partial products involving a bit equal to zero do not need to be performed or taken into account.
- For that matter, there are several filter designs in the literature that attempt to represent all filter coefficients as the sum of a minimum number of nonzero bits.

Serial multiplier

- Let A and B be two numbers of m and n bits, respectively, that can be represented using two's-complement arithmetic as

$$[A]_{2c} = s_A.a_1 a_2 \cdots a_m \tag{24}$$

$$[B]_{2c} = s_B.b_1 b_2 \cdots b_n \tag{25}$$

such that, from equation (20), A and B are given by

$$A = -s_A + a_1 2^{-1} + a_2 2^{-2} + \cdots + a_m 2^{-m} \tag{26}$$

$$B = -s_B + b_1 2^{-1} + b_2 2^{-2} + \cdots + b_n 2^{-n} \tag{27}$$

Serial multiplier

- Using two's-complement arithmetic, the product $P = AB$ is given by

$$\begin{aligned} P &= (-s_A + a_1 2^{-1} + \cdots + a_m 2^{-m})(-s_B + b_1 2^{-1} + \cdots + b_n 2^{-n}) \\ &= (-s_A + a_1 2^{-1} + \cdots + a_m 2^{-m}) b_n 2^{-n} \\ &\quad + (-s_A + a_1 2^{-1} + \cdots + a_m 2^{-m}) b_{n-1} 2^{-n+1} \\ &\quad \vdots \\ &\quad + (-s_A + a_1 2^{-1} + \cdots + a_m 2^{-m}) b_1 2^{-1} \\ &\quad - (-s_A + a_1 2^{-1} + \cdots + a_m 2^{-m}) s_B \end{aligned} \tag{28}$$

Serial multiplier

- We can perform this multiplication in a step-by-step manner by first summing the terms multiplied by b_n and b_{n-1}, taking the result and adding to the term multiplied by b_{n-2}, and so on.
- Let us develop this reasoning a bit further: The sum of the first two terms can be written as

$$\begin{aligned} C &= b_n 2^{-n} A + b_{n-1} 2^{-n+1} A \\ &= (-s_A + a_1 2^{-1} + \cdots + a_m 2^{-m}) b_n 2^{-n} \\ &\quad + (-s_A + a_1 2^{-1} + \cdots + a_m 2^{-m}) b_{n-1} 2^{-n+1} \\ &= 2^{-n+1} [b_n 2^{-1} (-s_A + a_1 2^{-1} + \cdots + a_m 2^{-m}) \\ &\quad + b_{n-1} (-s_A + a_1 2^{-1} + \cdots + a_m 2^{-m})] \end{aligned} \tag{29}$$

Serial multiplier

- From equation (22), the above equation becomes

$$C = 2^{-n+1}\left[b_n(-s_A + s_A 2^{-1} + a_1 2^{-2} + \cdots + a_m 2^{-m-1}) + b_{n-1}(-s_A + a_1 2^{-1} + \cdots + a_m 2^{-m})\right] \quad (30)$$

which can be represented in the form of the multiplication algorithm as

$$\begin{array}{cccccccccc} & (& & s_A & s_A & a_1 & \ldots & a_{m-1} & a_m &) \times b_n \\ + & (& & s_A & a_1 & a_2 & \ldots & a_m & &) \times b_{n-1} \\ \hline & & s_C & c_1 & c_2 & c_3 & \ldots & c_{m+1} & c_{m+2} & \end{array}$$

- And then

$$C = 2^{-n+2}(-s_C + c_1 2^{-1} + c_2 2^{-2} + c_3 2^{-3} + \cdots + c_{m+1} 2^{-m-1} + c_{m+2} 2^{-m-2}) \quad (31)$$

Serial multiplier

- Note that the sum of two positive numbers is always positive, and the sum of two negative numbers is always negative.
- Therefore, $s_C = s_A$. In fact, since in two's-complement arithmetic the sign has to be extended, the more compact representation for the summation above would be

$$
\begin{array}{cccccccccccc}
 & (& \ldots & s_A & s_A & s_A & s_A & a_1 & \ldots & a_{m-1} & a_m &) \times b_n \\
+ & (& \ldots & s_A & s_A & s_A & a_1 & a_2 & \ldots & a_m & &) \times b_{n-1} \\
\hline
 & & \ldots & s_A & s_A & c_1 & c_2 & c_3 & \ldots & c_{m+1} & c_{m+2} &
\end{array}
$$

Serial multiplier

- In the next step, C should be added to $b_{n-2}2^{-n+2}A$. This yields

$$\begin{aligned} D &= (-s_A + c_1 2^{-1} + \cdots + c_{m+2}2^{-m-2})2^{-n+2} \\ &\quad +(-s_A + a_1 2^{-1} + \cdots + a_m 2^{-m})b_{n-2}2^{-n+2} \end{aligned} \tag{32}$$

which can be represented in the compact form of the multiplication algorithm as

$$\begin{array}{cccccccccccc} & (& \ldots & s_A & s_A & c_1 & c_2 & \ldots & c_m & c_{m+1} & c_{m+2} &) \\ + & (& \ldots & s_A & s_A & a_1 & a_2 & \ldots & a_m & & &) \quad \times \quad b_{n-2} \\ \hline & & \ldots & s_A & d_1 & d_2 & d_3 & \ldots & d_{m+1} & d_{m+2} & d_{m+3} & \end{array}$$

- This is equivalent to

$$D = 2^{-n+3}(-s_A + d_1 2^{-1} + d_2 2^{-2} + \cdots + d_{m+2}2^{-m-2} + d_{m+3}2^{-m-3}) \tag{33}$$

Serial multiplier

- This process goes on until we obtain the next-to-last partial sum, Y.
- If B is positive, s_B is zero, and the final product, Z, is equal to Y, that is

$$Z = Y = -s_A + y_1 2^{-1} + \cdots + y_m 2^{-m} + y_{m+1} 2^{-m-1} + \cdots + y_{m+n} 2^{-m-n} \tag{34}$$

such that the two's complement of Z is given by

$$[Z]_{2c} = s_A.z_1 z_2 \cdots z_m z_{m+1} \cdots z_{m+n} = s_A.y_1 y_2 \cdots y_m y_{m+1} \cdots y_{m+n} \tag{35}$$

- On the other hand, if B is negative, then $s_B = 1$, and Y still needs to be subtracted from $s_B A$.

Serial multiplier

- This can be represented as

$$
\begin{array}{cccccccccc}
 & (& s_A & y_1 & y_2 & \dots & y_m & y_{m+1} & \dots & y_{m+n} &) & \\
- & (& s_A & a_1 & a_2 & \dots & a_m & & & &) & \times \quad s_B \\
\hline
 & & s_Z & z_1 & z_2 & \dots & z_m & z_{m+1} & \dots & z_{m+n} & &
\end{array}
$$

- The full precision two's-complement multiplication of A, with $(m+1)$ bits, by B, with $(n+1)$ bits, should be represented with $(m+n+1)$ bits.

Serial multiplier

- If we want to represent the final result using just the same number of bits as A, that is, $(m+1)$, we can use either rounding or truncation.
- For truncation, it suffices to disregard the bits $z_{m+1}, z_{m+2}, \ldots, z_{m+n}$.
- For rounding, we must add to the result, prior to truncation, a value equal to $\Delta = 2^{-m-1}$, such that

$$[\Delta]_{2c} = 0.\underbrace{0\cdots0}_{m \text{ zeros}}1 \tag{36}$$

Serial multiplier

- By looking at the algorithmic representation of the last partial sum above, one sees that to add Δ to the result is equivalent to summing bit 1 at position $(m+1)$, which is the nth position from the rightmost bit, z_{m+n}.
- Since it does not matter whether this bit is summed in the last or the first partial sum, it suffices to sum this bit during the first partial sum.
- Then the rounding can be performed by summing the number

$$[Q]_{2c} = 1.\underbrace{0\cdots0}_{n-1 \text{ zeros}} \tag{37}$$

to the first partial sum. In addition, since only $(m+1)$ bits of the product will be kept, we need to keep, from each partial sum, its $(m+1)$ most significant bits.

Serial multiplier

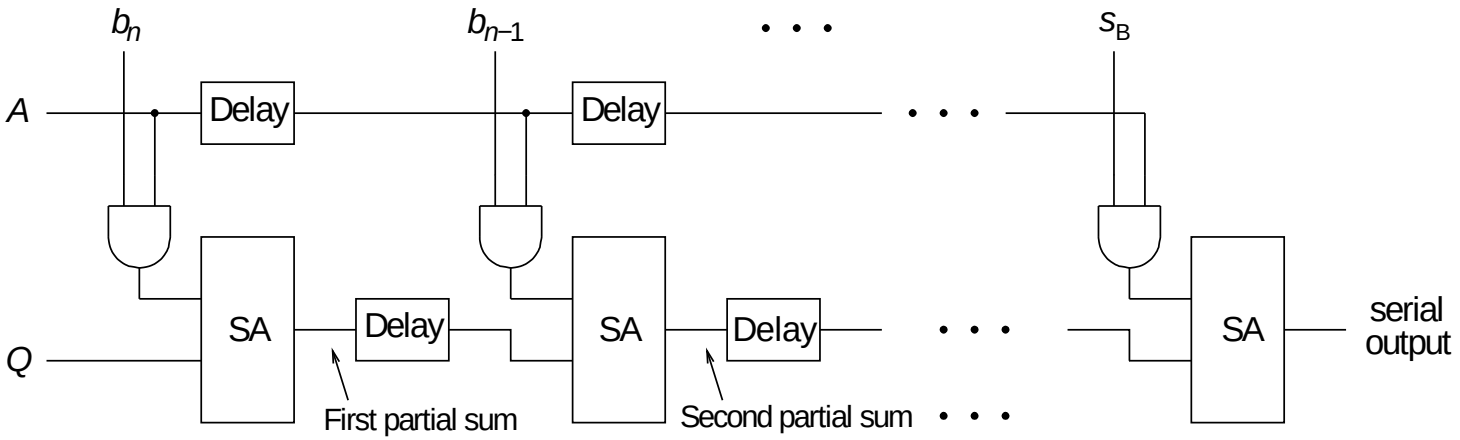


Figure 3: Schematic representation of a serial multiplier.

Serial multiplier

- The main idea behind a serial multiplier is to perform each partial sum using a serial adder such as the one depicted in Figure 1, taking care to introduce proper delays between the serial adder (SA) blocks, to align each partial sum properly with $b_j A$.
- This scheme is depicted in Figure 3, where the rounding signal Q is introduced in the first partial sum, and the least significant bits of A and Q are input first.
- In the scheme of Figure 3, depending on the values of the delay elements, one serial adder can begin to perform one partial sum as soon as the first bit of the previous partial sum becomes available.
- When this happens, we have what is called a pipelined architecture. The main idea behind the concept of pipelining is to use delay elements at strategic points of the system that allow one operation to start before the previous operation has finished.

Serial multiplier

- Figure 4 shows a pipelined serial multiplier. In this multiplier, the input A is in two's-complement format, while the input B is assumed to be positive.
- In this figure, the cells labeled D are D-type flip-flops, all sharing the same clock line, which is omitted for convenience, and the cells labeled SA represent the serial adder depicted in Figure 2a.
- The latch element used is shown in Figure 5. In this case, if the `enable` signal is high, after the clock, the output y becomes equal to the input x; otherwise, the output y keeps its previous value.
- In Figure 4, since B is assumed to be positive, then $s_B = 0$. Also, the rounding signal Q is such that

$$[Q]_{2c} = \begin{cases} 00\cdots00, & \text{for truncation} \\ \cdots01\ \underbrace{0\cdots0}_{n+1 \text{ zeros}}, & \text{for rounding} \end{cases} \tag{38}$$

Serial multiplier

- In the case of rounding, it is important to note that $[\mathrm{Q}]_{2c}$ has two more zeros to the right than the one in equation (37). This is so because this signal must be synchronized with input $\mathrm{b_n A}$, which is delayed by two bits before entering the serial adder of the first cell.

- Finally, the control signal CT is equal to

$$\mathrm{CT} = 00\underbrace{1\cdots1}_{\mathrm{m\ ones}}0 \tag{39}$$

- It should be noticed that all signals A, B, Q, CT should be input from right to left, that is, starting from their least-significant bits.

- Naturally, the output signal is generated serially, also starting from its least-significant bit, the first bit coming out after $(2\mathrm{n}+3)$ clock cycles and the entire word taking a total of $(2\mathrm{n}+\mathrm{m}+3)$ cycles to be calculated.

Serial multiplier

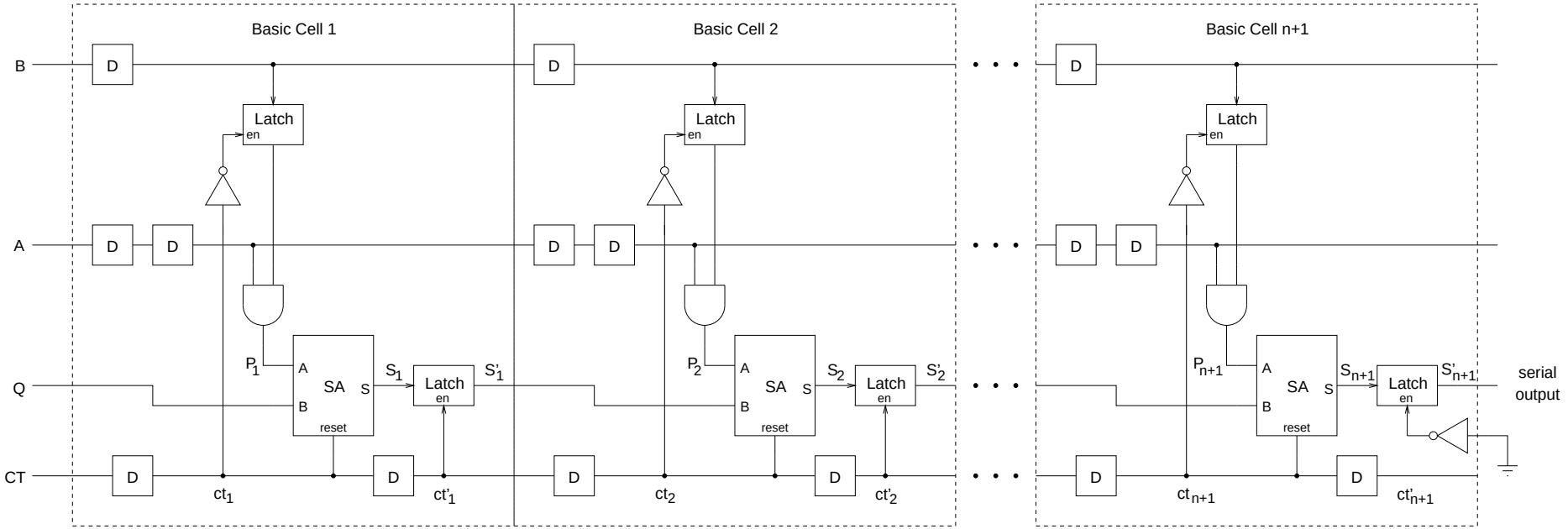


Figure 4: Basic architecture of the pipelined serial multiplier. Clock lines have been suppressed for convenience.

Serial multiplier

- In the literature, it has been shown that the multiplication of two words of lengths $(m+1)$ and $(n+1)$ using a serial or serial-parallel multiplier takes at least $(m+n+2)$ clock cycles, even in the cases where the final result is quantized with $(m+1)$ bits.
- Using a pipelined architecture, it is possible to determine the $(m+1)$-bit quantized product at every $(m+1)$ clock cycles.
- In the serial multiplier of Figure 4, each basic cell performs one partial sum of the multiplication algorithm. As the result of cell j is being generated, it is directly fed to cell $(j+1)$, indicating the pipelined nature of the multiplier.

Serial multiplier

- A detailed explanation of the behavior of the circuit is given below:
  - (i) After the $(2j+1)$th clock cycle, b_{m-j} will be at the input of the upper latch of cell $j+1$. The first 0 of the control signal CT will be at input ct_{j+1} at this time, which will reset the serial adder and enable the upper latch of cell $j+1$. Therefore, after the $(2j+2)$th clock cycle, b_{m-j} will be at the output of the upper latch, and will remain there for m clock cycles (number of 1s of CT).
  - (ii) After the $(2j+2)$th clock cycle, a_m will be at the input of the AND gate of cell $j+1$, and therefore $a_m b_{m-j}$ will be the P_{j+1} input of the serial adder. Since at this time the first 0 of the control signal CT will be at input ct'_{j+1}, the lower latch becomes on hold. Therefore, although the first bit of the $(j+1)$th partial sum is at S_{j+1} after the $(2j+2)$th clock cycle, it will not be at S'_{j+1} after the $(2j+3)$th clock cycle, and it will be discarded. Since in the next $m+1$ clock cycles the bit 1 will be at ct'_{j+1}, the remaining $m+1$ bits of the $(j+1)$th partial sum will pass from S_{j+1} to S'_{j+1}, which is input to the next basic cell.

Serial multiplier

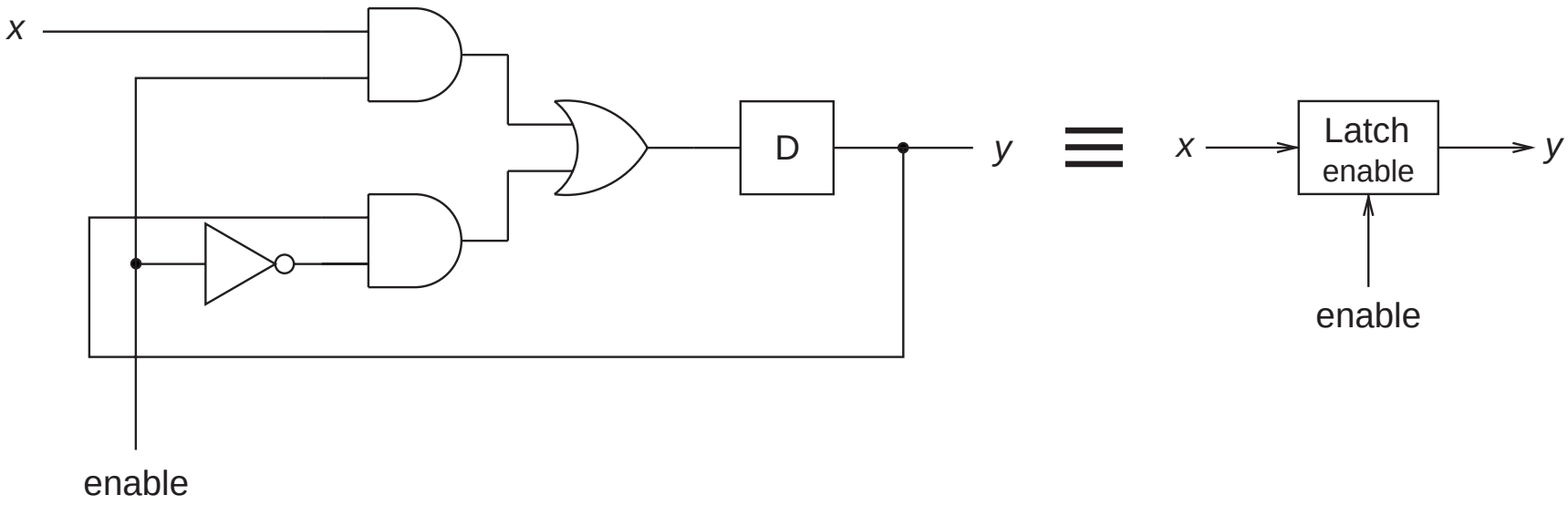


Figure 5: Latch element.

Serial multiplier

- (cont.)
  - (iii) After the $(2j + 2 + k)$th clock cycle, $a_{m-k} b_{m-j}$ will be the P_{j+1} input of the serial adder of cell $j + 1$. Therefore, after the $(2j + 2 + m)$th clock cycle, $s_A b_{m-j}$ will be at the P_{j+1} input of the serial adder of cell $j + 1$. This is equivalent to saying that the last bit of the $(j + 1)$th partial sum will be at the input of the lower latch of cell $j + 1$ at this time. Since then ct'_{j+1} is still 1, then, after the $(2j + 3 + m)$th clock cycle the last bit of the partial sum of cell $j + 1$ will be at the output of the lower latch of cell $j + 1$. But from this time on, ct'_{j+1} will be zero, and therefore the output of the latch of cell $j + 1$ will hold the last bit of the $(j + 1)$th partial sum. Since this last bit represents the sign of the $(j + 1)$th partial sum, it performs the sign extension necessary in two's-complement arithmetic.

Serial multiplier

- (cont.)
  - (iv) Since there is no need to perform sign extension of the last partial sum, the lower latch of the last cell is always enabled, as indicated in Figure 4.
  - (v) Since each basic cell but the last one only outputs m bits apart from the sign extensions, then the serial output at the last cell will contain the product either truncated or rounded to $m + 1$ bits, depending on the signal Q.

Example 11.2

- Verify how the pipelined serial multiplier depicted in Figure 4 processes the product of the binary numbers A = 1.1001 and B = 0.011.

Example 11.2 - Solution

- We have that $m = 4$ and $n = 3$. Therefore, the serial multiplier has four basic cells.
- We expect the least-significant bit of the truncated product to be output after $(2n + 3) = 9$ clock cycles and its last bit after $(2n + m + 3) = 13$ clock cycles.
- Supposing that the quantization signal Q is zero, which corresponds to a truncated result, we have that (the variable t indicates the clock cycle after which the values of the signals are given; $t = 0$ means the time just before the first clock pulse):

Example 11.2 - Solution

t	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P_1	0	0	0	0	0	0	0	1	1	0	0	1	0	0
S_1	0	0	0	0	0	0	0	1	1	0	0	1	0	0
ct'_1	0	0	0	0	0	0	0	1	1	1	1	0	0	0
S'_1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
P_2	0	0	0	0	0	1	1	0	0	1	0	0	0	0
S_2	0	0	0	0	0	1	0	1	0	1	0	0	0	0
ct'_2	0	0	0	0	0	1	1	1	1	0	0	0	0	0
S'_2	1	1	1	1	1	0	1	0	0	0	0	0	0	0
P_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S_3	1	1	1	1	1	0	1	0	0	0	0	0	0	0
ct'_3	0	0	0	1	1	1	1	0	0	0	0	0	0	0
S'_3	1	1	1	1	0	1	0	0	0	0	0	0	0	0
P_4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S_4	1	1	1	1	0	1	0	0	0	0	0	0	0	0
ct'_4	1	1	1	1	1	1	1	1	1	1	1	1	1	1
S'_4	**1**	**1**	**1**	**0**	**1**	0	0	0	0	0	0	0	0	0

- And the computed product is 1.1101.

Example 11.2 - Solution

- For rounding, since $n = 3$, we have, from equation (38), that $[Q]_{2c} = \cdots 0010000$. Thus, the operation of the serial multiplier is as follows:

t	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Q	0	0	0	0	0	0	0	0	0	1	0	0	0	0
P_1	0	0	0	0	0	0	0	1	1	0	0	1	0	0
S_1	0	0	0	0	0	0	0	1	1	1	0	1	0	0
ct'_1	0	0	0	0	0	0	0	1	1	1	1	0	0	0
S'_1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
P_2	0	0	0	0	0	1	1	0	0	1	0	0	0	0
S_2	0	0	0	0	0	1	0	1	1	1	0	0	0	0
ct'_2	0	0	0	0	0	1	1	1	1	0	0	0	0	0
S'_2	1	1	1	1	1	0	1	1	0	0	0	0	0	0
P_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S_3	1	1	1	1	1	0	1	1	0	0	0	0	0	0
ct'_3	0	0	0	1	1	1	1	0	0	0	0	0	0	0
S'_3	1	1	1	1	0	1	0	0	0	0	0	0	0	0
P_4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S_4	1	1	1	1	0	1	0	0	0	0	0	0	0	0
ct'_4	1	1	1	1	1	1	1	1	1	1	1	1	1	1
S'_4	**1**	**1**	**1**	**0**	**1**	0	0	0	0	0	0	0	0	0

- That is, the rounded product is also 1.1101.

Serial multiplier

- The general multiplier for two's-complement arithmetic (without the positive restriction in one of the factors) can be obtained from the multiplier seen in Figure 4 by a slight modification of the connections between the last two basic cells.
- In fact, from the representation given in equation (20), we note that the general multiplication of any two numbers represented in two's complement is equal to the multiplication of a positive number and a two's-complement number, except that in the last step we must perform the subtraction of the product between the data and the coefficient sign bit from the partial result obtained at that point.
- Then, we must perform a subtraction to obtain $S_{n+1} = S'_n - s_A A$.

Serial multiplier

- It can be shown that, using two's-complement arithmetic, $X - Y = \overline{Y + \overline{X}}$, where $\overline{X}$ represents the inversion of all bits of X.
- Therefore, in the last cell, we should invert S'_n before it is input to the $(n+1)$th serial adder, and then invert its output.
- Thus, the connections to the last basic cell should be modified as shown in Figure 6.

Serial multiplier

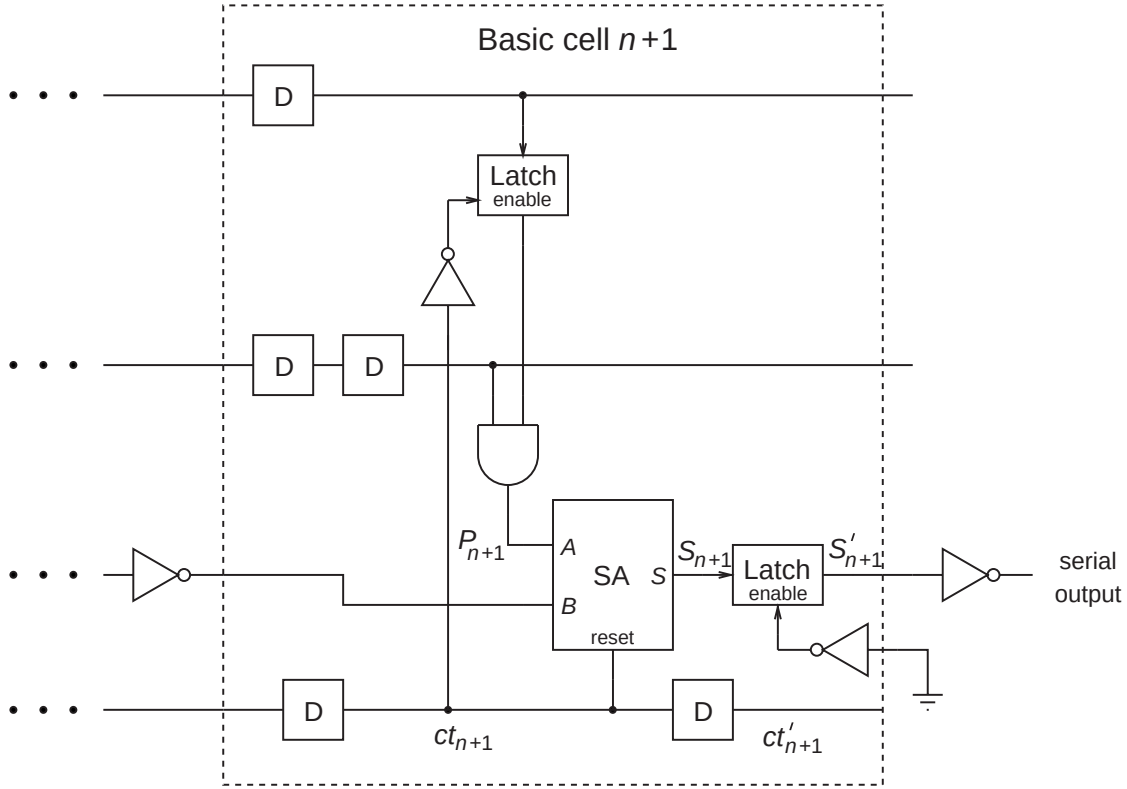


Figure 6: Tail-end of the modified general multiplier for two's-complement arithmetic.

Serial multiplier

- An alternative implementation for the rounding quantization, instead of using the Q signal, consists of forcing the carry signal in serial adder of the nth basic cell to 1 while it is performing the addition of $a_m b_1$ to the $(n-1)$th partial sum.
- So far, we have focused on single-precision multipliers, that is, the final product is quantized, either by rounding or truncation.
- As seen in previous chapters, in many cases it is desirable that the final product presents double precision to avoid, for instance, nonlinear oscillations.
- Such operations can be easily performed with the basic multipliers seen so far, artificially doubling the precision of the two multiplying factors by juxtaposing the adequate number of bits 0 to the right of each input number.

Serial multiplier

- For example, the exact multiplication of two inputs with $(n+1)$ bits each is obtained by doubling the number of bits in each input by adding $(n+1)$ bits 0 to the right of each number.
- Naturally, in this case, we need $(n+1)$ more basic cells, and the complete operation takes twice as long to be implemented as basic single-precision multiplication.
- For binary representations other than the two's complement, the implementation of the corresponding serial multipliers can be found in the associated literature.

Parallel adder

- Parallel adders can be easily constructed by interconnecting several full adders as shown in Figure 7, where we can observe that the carry signal in each cell propagates to the next cell through the overall adder structure.
- The main time-consuming operation in this realization is the propagation of the carry signal that must go through all the full adders to form the desired sum.
- This problem can be reduced at the cost of an increase in the hardware complexity.

Parallel adder

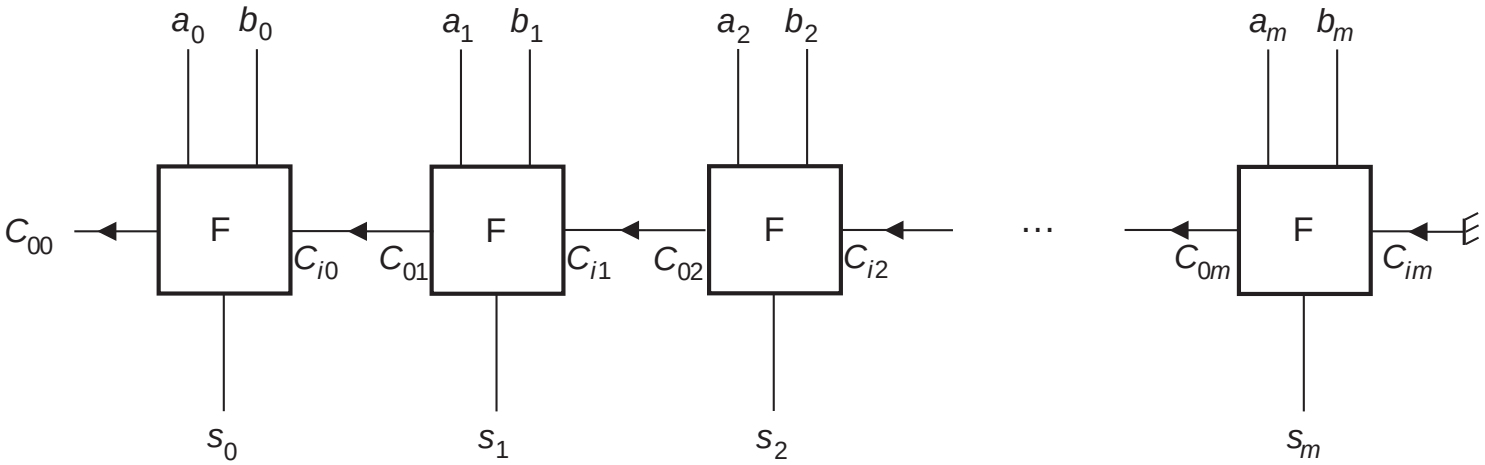


Figure 7: Block diagram of the parallel adder.

Parallel multiplier

- A parallel multiplier is usually implemented as a matrix of basic cells, in which the internal data propagates horizontally, vertically, and diagonally in an efficient and ordered way.
- In general, such multipliers occupy a very large area of silicon and consume significant power.
- For these reasons they are only used in cases where time is a major factor in the overall performance of the given digital processing system, as in major DSP chips of today.
- Since a digital filter is basically composed of delays, adders, and multipliers (using serial or parallel arithmetic), we now have all the tools necessary to implement a digital filter.

Parallel multiplier

- The implementation is carried out by properly interconnecting such elements according to a certain realization, which should be chosen from among the ones seen in the previous chapters.
- If in the resulting design, the sampling rate multiplied by the number of bits used to represent the signals is below the reachable processing speed, multiplexing techniques can be incorporated to optimize the use of hardware resources.

Parallel multiplier

- There are two main multiplexing approaches.
- In the first one, a single filter processes several input signals that are appropriately multiplexed.
- In the second one, the filter coefficients are multiplexed, resulting in several distinct transfer functions.
- It is always possible to combine both approaches, if desired, as suggested in Figure 8, where a memory element stores several sets of coefficients corresponding to different digital filters to be multiplexed.

Parallel multiplier

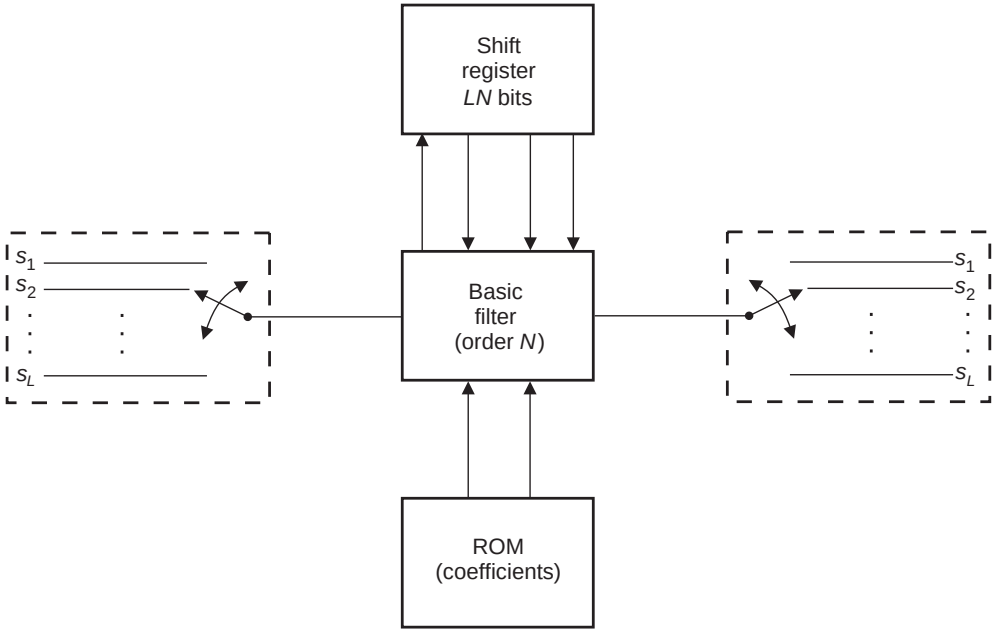


Figure 8: Fully multiplexed realization of digital filters.

Distributed arithmetic implementation

- An alternative approach for implementing digital filters, the so-called distributed arithmetic, avoids the explicit implementation of the multiplier element.
- Such a concept first appeared in the open literature in 1974, although a patent describing it had been issued in December 1973.
- The basic idea behind the concept of distributed arithmetic is to perform the summation of the products between filter coefficients and internal signals without using multipliers.
- For example, suppose one wishes to calculate the following expression:

$$y = \sum_{i=1}^{N} c_i X_i \tag{40}$$

where c_i are the filter coefficients and X_i are a set of signals represented in two's complement with $(b+1)$ bits.

Distributed arithmetic implementation

- Assuming that the X_i are properly scaled such that $|X_i| < 1$, we can rewrite equation (40) as

$$y = \sum_{i=1}^{N} c_i \left(\sum_{j=1}^{b} x_{i_j} 2^{-j} - x_{i_0} \right) \tag{41}$$

  where x_{i_j} corresponds to the jth bit of x_i, and x_{i_0} to its sign.

- By reversing the summation order in this equation, the following relationship applies

$$y = \sum_{j=1}^{b} \left(\sum_{i=1}^{N} c_i x_{i_j} \right) 2^{-j} - \sum_{i=1}^{N} c_i x_{i_0} \tag{42}$$

Distributed arithmetic implementation

- If we define the function $s(\cdot)$ of N binary variables $z_1, z_2, \ldots, z_N$, as

$$s(z_1, z_2, \ldots, z_N) = \sum_{i=1}^{N} c_i z_i \tag{43}$$

then equation (42) can be written as

$$y = \sum_{j=1}^{b} s(x_{1_j}, x_{2_j}, \ldots, x_{N_j}) 2^{-j} - s(x_{1_0}, x_{2_0}, \ldots, x_{N_0}) \tag{44}$$

- Using the notation $s_j = s(x_{1_j}, x_{2_j}, \ldots, x_{N_j})$, then the above equation can be written as

$$y = \{\ldots[(s_b 2^{-1} + s_{b-1})2^{-1} + s_{b-2}]2^{-1} + \cdots + s_1\}2^{-1} - s_0 \tag{45}$$

Distributed arithmetic implementation

- The value of s_j depends on the jth bit of all signals that determine y. Equation (45) gives a methodology to compute y: we first compute s_b, then divide the result by 2 with a right-shift operation, add the result to s_{b-1}, then divide the result by 2, add it to s_{b-2}, and so on. In the last step, s_0 must be subtracted from the last partial result.
- Overall, the function $s(\cdot)$ in equation (43) can assume at most 2^N possible distinct values, since all of its N variables are binary.
- Thus, an efficient way to compute $s(\cdot)$ is to pre-determine all its possible values, which depend on the values of the coefficients c_i, and store them in a memory unit whose addressing logic must be based on the synchronized data content.

Distributed arithmetic implementation

- The distributed arithmetic implementation of equation (40) is as shown in Figure 9.
- In this implementation, the words $X_1, X_2, \ldots, X_N$ are stored in shift-registers SR_1, $SR_2, \ldots, SR_N$, respectively, each one with $(b+1)$ bits.
- The N outputs of the shift-registers are used to address a ROM unit.
- Then, once the words are loaded in the shift-registers, after the jth right-shift, the ROM will be addressed with $x_{1\,b-j}\, x_{2\,b-j} \cdots x_{N\,b-j}$, and the ROM will output s_{b-j}.

Distributed arithmetic implementation

- This value is then loaded into register A to be added to the partial result from another register, B, which stores the previous accumulated value.
- The result is divided by 2 (see equation (45)).
- For each calculation, register A is initialized with s_b and register B is reset to contain only zeros.
- Register C is used to store the final sum, obtained from the subtraction of s_0 from the next-to-last sum.
- Naturally, all the above calculations could have been implemented without register A, whose importance lies in the possibility of accessing the memory unit simultaneously to the adder/subtractor operation, thus forming a pipelined architecture.

Distributed arithmetic implementation

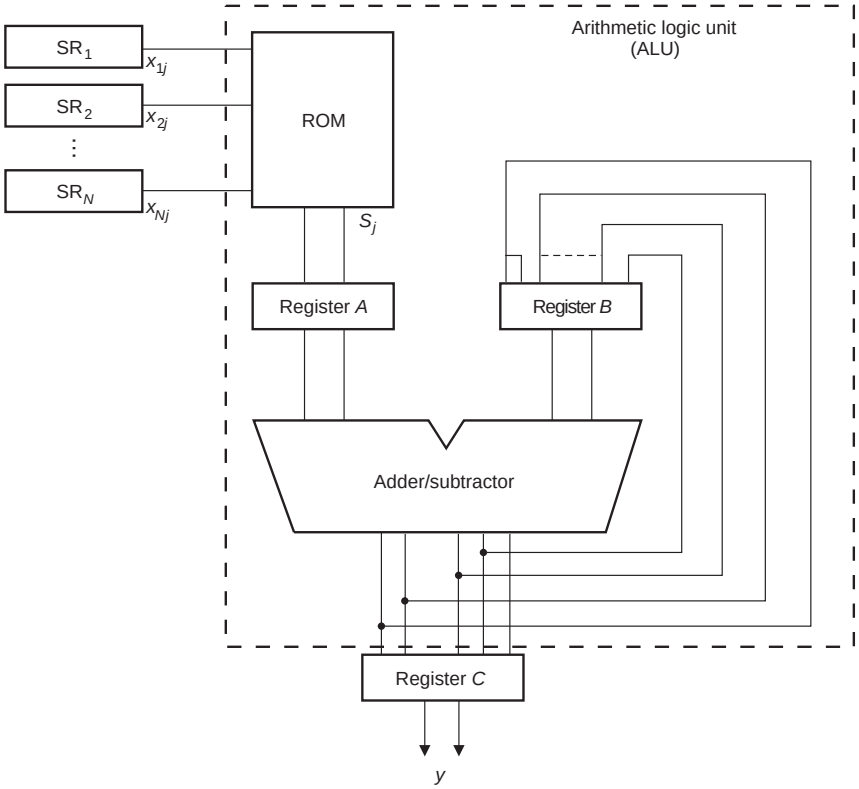


Figure 9: Basic architecture for implementing equation (40) using distributed arithmetic.

Distributed arithmetic implementation

- Using the implementation illustrated in Figure 9, the complete time interval for computing one sample of the output y entails $(b + 1)$ clock cycles, and therefore depends solely on the number of bits of each word, and is not a function of the number, N, of partial products involved to form y.
- The duration of the clock cycle, in this case, is determined by the maximum value between the times of a memory access and a computation of an addition or subtraction operation.
- The number of bits, b, used to represent each word greatly affects the memory size, along with the number, N, of partial sums, which should not be made too large in order to limit the memory access time.

Distributed arithmetic implementation

- The value of b depends basically on the dynamic range necessary to represent $|s_j|$ and the precision required to represent the coefficients c_i to avoid large coefficient quantization effects.
- We can greatly increase the overall processing speed for computing y as in equation (40), by reading the desired values of s_j in parallel, and using several adders in parallel to determine all necessary partial sums.
- This distributed arithmetic can be used to implement several of the digital filter structures presented in this book.

Distributed arithmetic implementation

- For instance, the second-order direct-form realization implementing the following transfer function

$$H(z) = \frac{b_0 z^2 + b_1 z + b_2}{z^2 + a_1 z + a_2} \tag{46}$$

which corresponds to the following difference equation

$$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) - a_1 y(n-1) - a_2 y(n-2) \tag{47}$$

that can be mapped onto the implementation seen in Figure 10, where the present and past values of the input signal are used in conjunction with the past values of the output signal to address the memory unit.

Distributed arithmetic implementation

- In other words, we make $X_i = x(n-i)$, for $i = 0, 1, 2$, $X_3 = y(n-1)$, and $X_4 = y(n-2)$. The function $s(\cdot)$, which determines the content of the memory unit, is then given by

$$s(z_1, z_2, z_3, z_4, z_5) = b_0 z_1 + b_1 z_2 + b_2 z_3 - a_1 z_4 - a_2 z_5 \qquad (48)$$

- Therefore, at a given instant, the quantity s_j is

$$s_j = b_0 x_j(n) + b_1 x_j(n-1) + b_2 x_j(n-2) - a_1 y_j(n-1) - a_2 y_j(n-2) \qquad (49)$$

- As the number of partial sums required to calculate y is $N = 5$, the memory unit in this example should have 32 positions of L bits, where L is the number of bits established to represent s_j.

Distributed arithmetic implementation

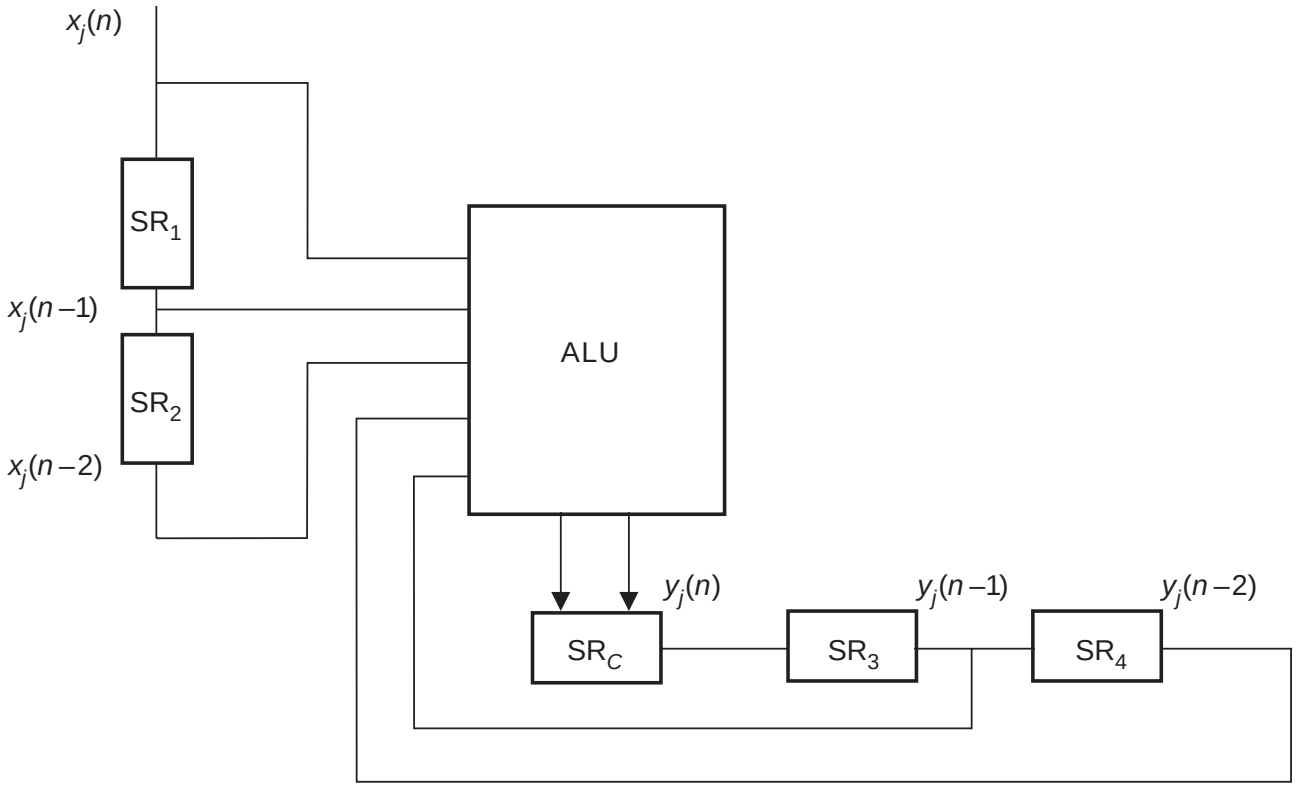


Figure 10: Second-order direct-form filter implemented with distributed arithmetic.

Distributed arithmetic implementation

- In general, all coefficients b_i and a_i in equation (48) should be already quantized.
- This is because it is easier to predict the quantization effects at the stage of filter design, following the theory presented later in this chapter, than to analyze such effects after quantizing s_j as given in equation (48).
- In fact, performing the quantization on s_j can even introduce nonlinear oscillations at the output of a given structure which was initially shown to be immune to limit cycles.
- Limit cycles are dealt with in Section 11.8, and such quantization effect is illustrated in Example 11.13.

Distributed arithmetic implementation

- For the second-order state-variable realization (as given in Figure 13.5 of the book), the distributed arithmetic implementation is shown in Figure 11.
- In this case, the contents of the memory units are generated by

$$s_{1j} = a_{11}x_{1j}(n) + a_{12}x_{2j}(n) + b_1x_j(n); \text{ for the ROM of the ALU}_1 \quad (50)$$

$$s_{2j} = a_{22}x_{2j}(n) + a_{21}x_{1j}(n) + b_2x_j(n); \text{ for the ROM of the ALU}_2 \quad (51)$$

$$s_{3j} = c_1x_{1j}(n) + c_2x_{2j}(n) + dx_j(n); \text{ for the ROM of the ALU}_3 \quad (52)$$

- Each memory unit has $8 \times L$ words, where L is the established wordlength for the given s_{ij}.

Distributed arithmetic implementation

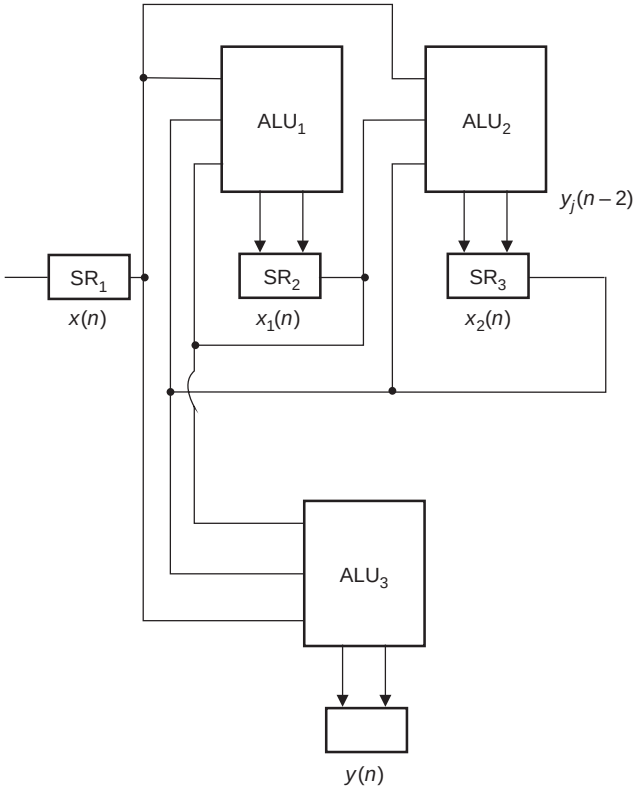


Figure 11: Second-order state-space filter implemented with distributed arithmetic.

Distributed arithmetic implementation

- The implementation of high-order filters using distributed arithmetic is simple when the parallel and cascade forms seen in Chapter 4 are used.
- Both forms can be implemented using a single block realization, whose coefficients are multiplexed in time to perform the computation of a specific second-order block.
- The cascade version of this type of implementation is represented in Figure 12. Such an approach is very efficient in terms of chip area and power consumption.
- Faster versions, however, are possible using both the parallel and cascade forms.
- For instance, the fast parallel approach is depicted in Figure 13, where all numerical calculations can be performed in parallel.
- The fast cascade form can be made more efficient if delay elements are added in between the blocks to allow pipelined processing.

Distributed arithmetic implementation

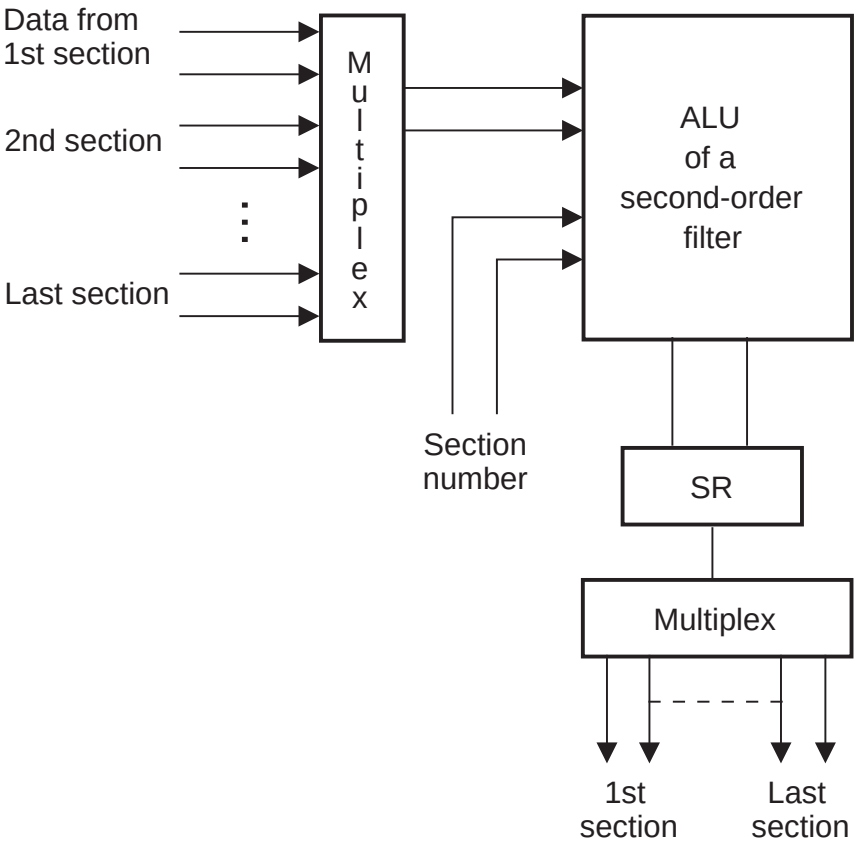


Figure 12: Implementation of the cascade form using distributed arithmetic.

Distributed arithmetic implementation

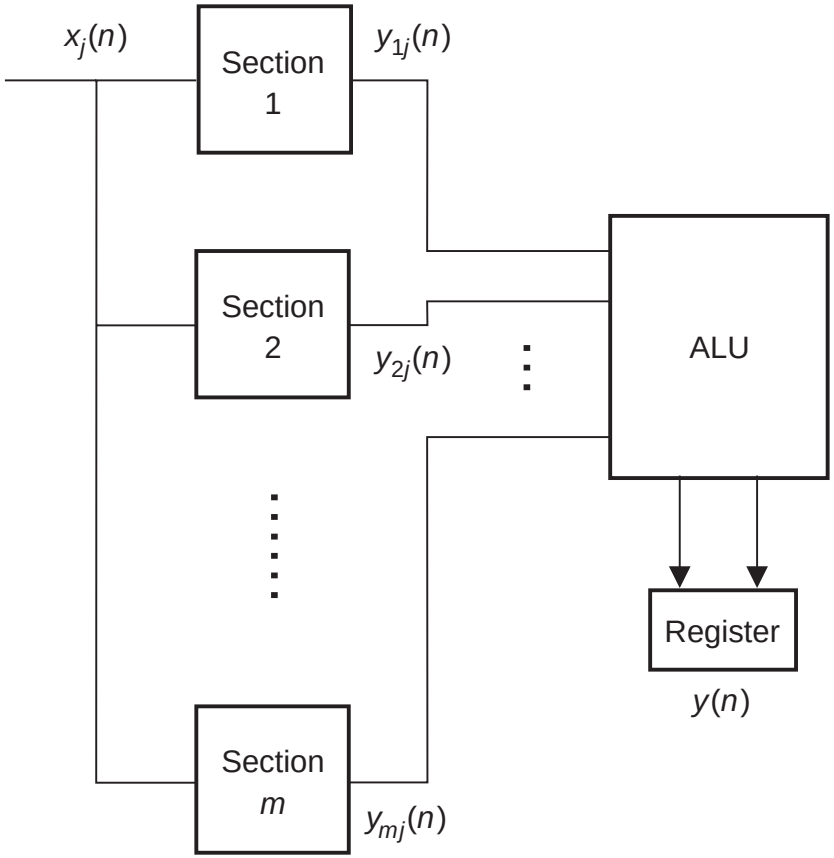


Figure 13: Fast implementation of the parallel form using distributed arithmetic.

Distributed arithmetic implementation

- The distributed arithmetic technique presented in this section can also be used to implement other digital filter structures, as well as some specific processors for the computation of the FFT.

Product quantization

- A finite-wordlength multiplier can be modeled in terms of an ideal multiplier followed by a single additive noise source $e(n)$, as shown in Figure 14.

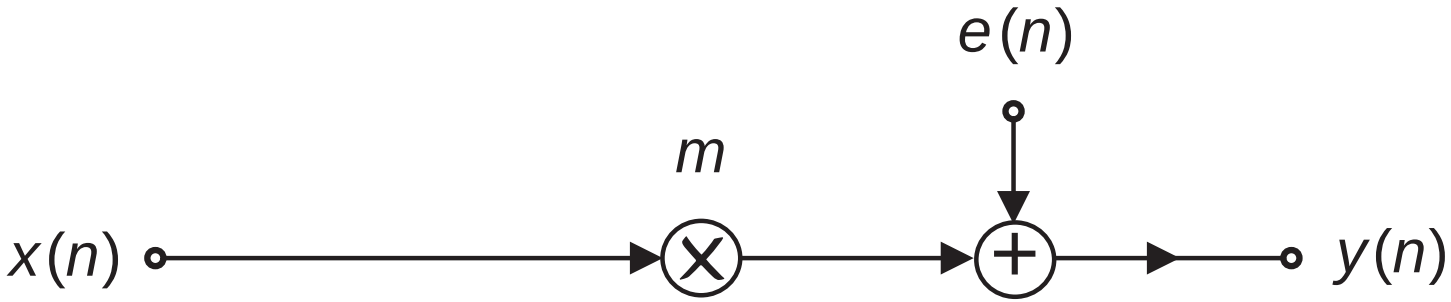


Figure 14: Noise model for multiplier.

Product quantization

- Three distinct approximation schemes can be employed after a multiplication, namely: rounding, truncation, and magnitude truncation. We analyze their effects in numbers represented in two's complement.
- Product quantization by rounding leads to a result in finite precision whose value is the closest possible to the actual value.
- If we assume that the dynamic range throughout the digital filter is much larger than the value of the least significant bit $q = 2^{-b}$ (b corresponds to n in equations (1)–(11)), the probability density function of the quantization error is depicted in Figure 15a.

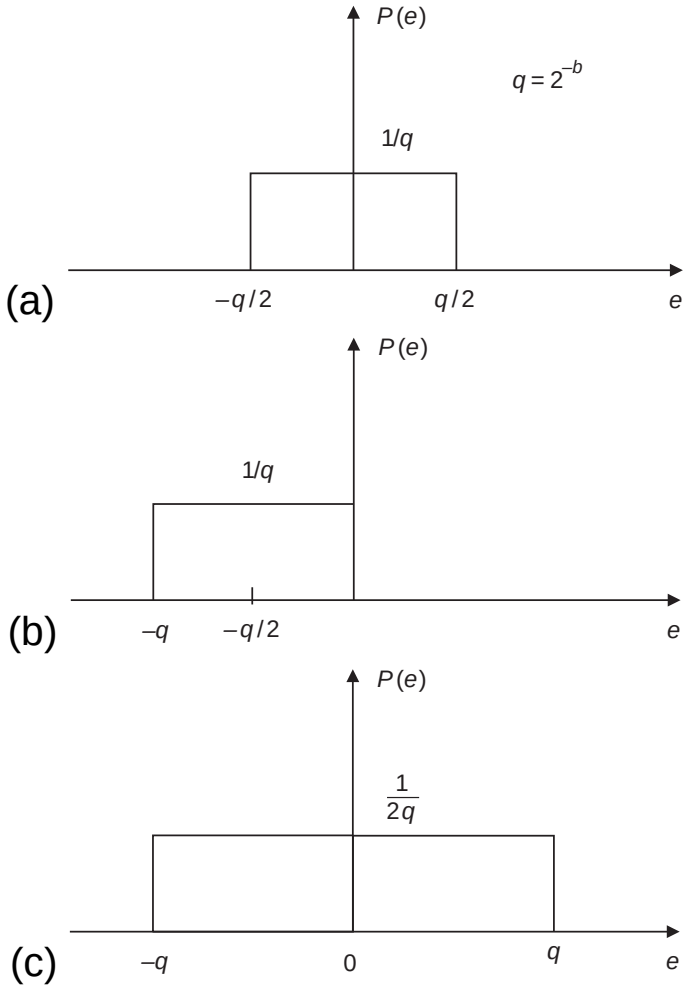


Figure 15: Probability density functions for the product-quantization error: (a) rounding; (b) truncation; (c) magnitude truncation.

Product quantization

- The mean or expected value of the quantization error due to rounding is zero, that is

$$E\{e(n)\} = 0 \tag{53}$$

- Also, it is easy to show that the variance of the noise $e(n)$ is given by

$$\sigma_e^2 = \frac{q^2}{12} = \frac{2^{-2b}}{12} \tag{54}$$

- In the case of truncation of a number, where the result is always less than the original value, the probability density function is as shown in Figure 15b, the expected value for the quantization error is

$$E\{e(n)\} = -\frac{q}{2} \tag{55}$$

and the error variance amounts to

$$\sigma_e^2 = \frac{2^{-2b}}{12} \tag{56}$$

Product quantization

- This type of quantization is not adequate, in general, because the errors associated with a nonzero mean value, although small, tend to propagate through the filter, and their effect can be sensed at the filter output.
- If we apply magnitude truncation, which necessarily implies reducing the magnitude of the number, the probability density function has the form depicted in Figure 15c.
- In this case, the mean value of the quantization error is

$$\mathrm{E}\{e(n)\} = 0 \tag{57}$$

and the variance is

$$\sigma_e^2 = \frac{2^{-2b}}{3} \tag{58}$$

Product quantization

- Due to these results, it is easy to understand why rounding is the most attractive form of quantization, since it generates the smallest noise variance while maintaining the mean value of the quantization error equal to zero.
- Magnitude truncation, besides having a noise variance four times greater than rounding, leads to a higher correlation between the signal and quantization noise (for example, when the signal is positive/negative, the quantization noise is also positive/negative), which is a strong disadvantage, as will soon be clear.
- However, the importance of magnitude truncation can not be overlooked, since it can eliminate granular limit cycles in recursive digital filters, as will be shown later in this chapter.
- At this point, it is interesting to study how the signal quantization affects the output signal.

Product quantization

- In order to simplify the analysis of roundoff-noise effects, the following assumptions are made regarding the filter internal signals:
  - Their amplitude is much larger than the quantization error.
  - Their amplitude is small enough that overflow never occurs.
  - They have a wide spectral content.

  These assumptions imply that:
  - (i) The quantization errors at different instants are uncorrelated, that is, $e_i(n)$ is uncorrelated with $e_i(n+l)$, for $l \neq 0$.
  - (ii) Errors at different nodes are uncorrelated, that is, $e_i(n)$ is uncorrelated with $e_j(n)$, for $i \neq j$.

Product quantization

- From the above considerations, the contributions of different noise sources can be accounted for separately and added to determine the overall roundoff error at the filter output.
- However, one should note that (i) and (ii) do not hold when magnitude truncation is used, due the inherent correlation between the signal and the quantization error.
- Therefore, one should bear in mind that, for the magnitude truncation scheme, the analysis that follows does not lead to accurate results.
- Denoting the error variance due to each internal signal quantization by σ_e^2, and assuming that the quantization error is the outcome of a white noise process, the power spectral density (PSD) of a given noise source is

$$\Gamma_E(e^{j\omega}) = \sigma_e^2 \tag{59}$$

Product quantization

- In Chapter 2, we showed that a linear system having transfer function $H(z)$, if a stationary signal with PSD $\Gamma_X(e^{j\omega})$ is input, then the PSD of the output is $\Gamma_Y(e^{j\omega}) = H(e^{j\omega})H(e^{-j\omega})\Gamma_X(e^{j\omega})$.

- Therefore, in a fixed-point digital-filter implementation, the PSD of the output noise $y(n)$ is given by

$$\Gamma_Y(e^{j\omega}) = \sigma_e^2 \sum_{i=1}^{K} G_i(e^{j\omega})G_i(e^{-j\omega}) \tag{60}$$

where K is the number of multipliers of the filter, and $G_i(e^{j\omega})$ is the frequency response associated to the transfer function $G_i(z)$ from each multiplier output, $g_i(n)$, to the output of the filter, as indicated in Figure 16.

Product quantization

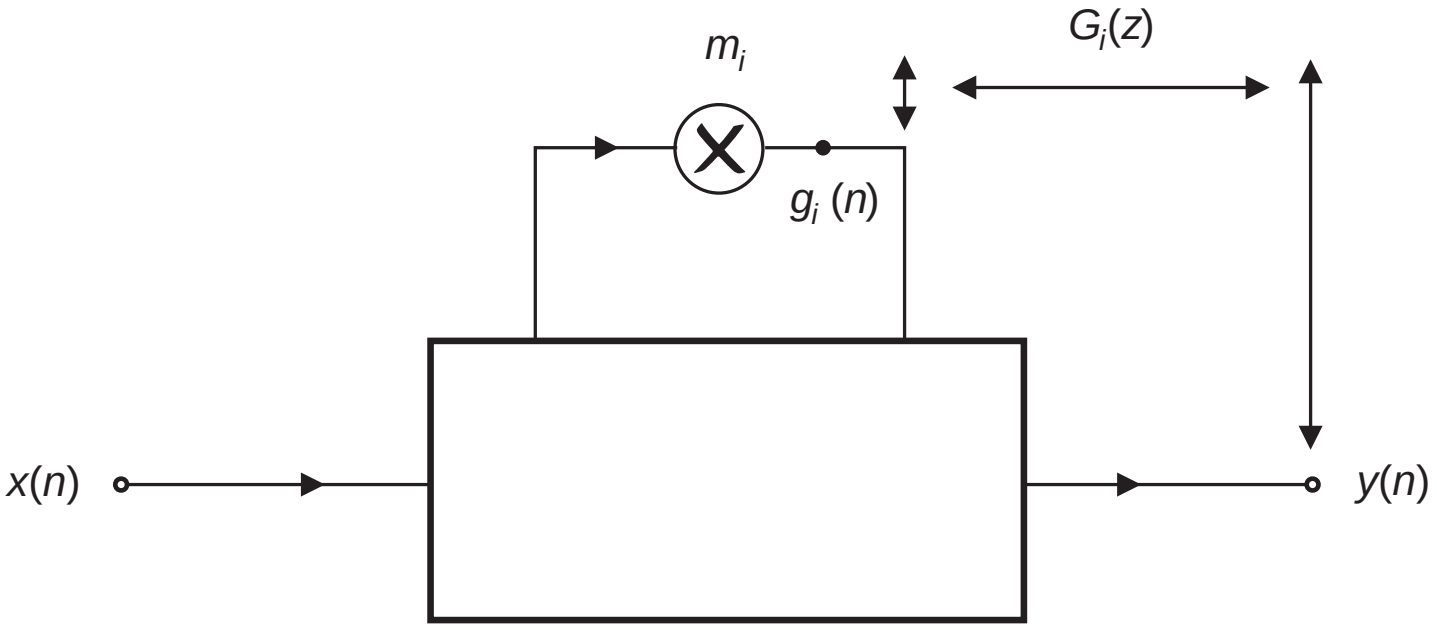


Figure 16: Noise transfer function for a digital filter.

Product quantization

- A common figure of merit for evaluating the performance of digital filters is the relative power spectral density (RPSD) of the output noise, defined in dB as

$$\text{RPSD} = 10 \log \frac{\Gamma_Y(e^{j\omega})}{\Gamma_E(e^{j\omega})} = 10 \log \sum_{i=1}^{K} \left| G_i(e^{j\omega}) \right|^2 \tag{61}$$

- This figure eliminates the dependence of the output noise on the filter wordlength, thus representing a true measure of the extent to which the output noise depends on the internal structure of the filter.

Product quantization

- A simpler but useful performance criterion to evaluate the roundoff noise generated in digital filters is the noise gain or the relative noise variance, defined by

$$\begin{aligned}\frac{\sigma_y^2}{\sigma_e^2} &= \frac{1}{\pi}\int_0^\pi \sum_{i=1}^{K} \left(G_i(z)G_i(z^{-1})\right)^2_{z=e^{j\omega}}\, d\omega \\ &= \frac{1}{\pi}\int_0^\pi \sum_{i=1}^{K} \left|G_i(e^{j\omega})\right|^2 d\omega \\ &= \frac{1}{\pi}\sum_{i=1}^{K}\int_0^\pi \left|G_i(e^{j\omega})\right|^2 d\omega \\ &= \sum_{i=1}^{K} \left\|G_i(e^{j\omega})\right\|_2^2 \end{aligned} \tag{62}$$

Product quantization

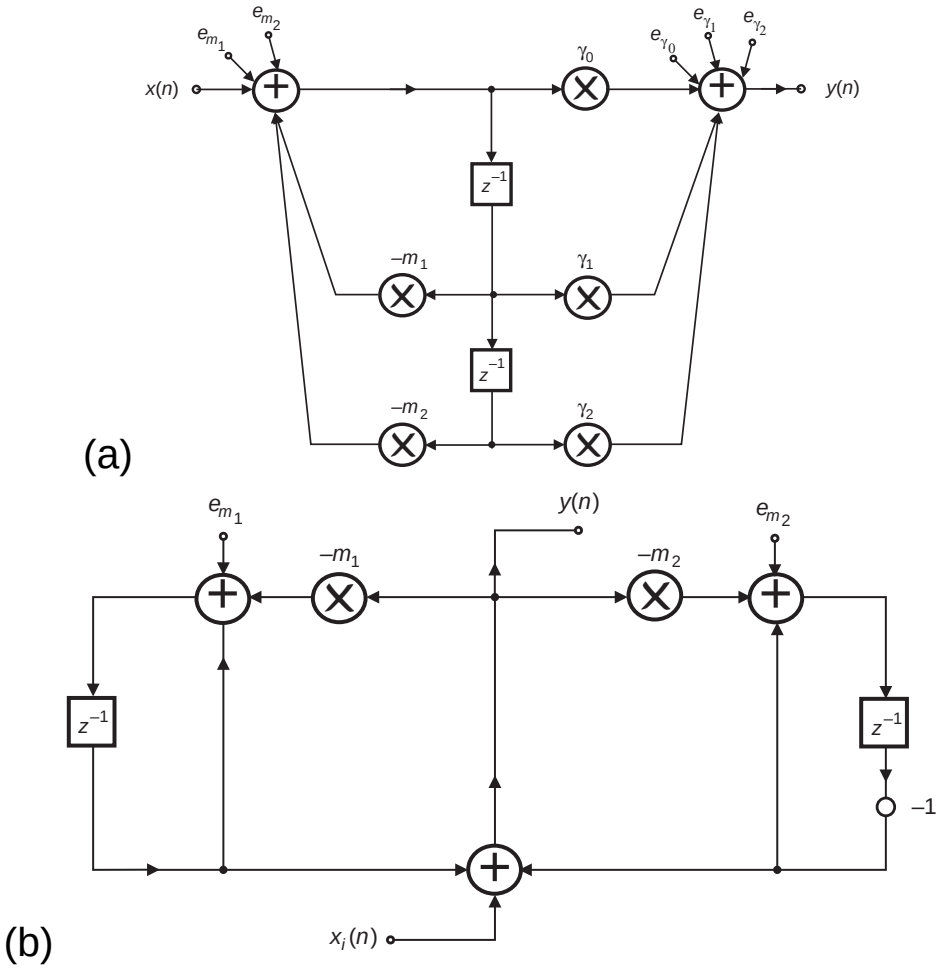


Figure 17: Second-order networks.

Product quantization

- Another noise source that must be accounted for in digital filters is the roundoff noise generated when the input-signal amplitude is quantized in the process of analog-to-digital conversion.
- Since the input-signal quantization is similar to product quantization, it can be represented by including a noise source at the input of the digital filter structure.

Example 11.3

- Compute the quantization noise PSD at the output of the networks shown in Figure 17, assuming that fixed-point arithmetic is employed.

Example 11.3 - Solution

- For the structure of Figure 17a, we have that

$$G_{m_1}(z) = G_{m_2}(z) = H(z) \tag{63}$$

$$G_{\gamma_0}(z) = G_{\gamma_1}(z) = G_{\gamma_2}(z) = 1 \tag{64}$$

where

$$H(z) = \frac{\gamma_0 z^2 + \gamma_1 z + \gamma_2}{z^2 + m_1 z + m_2} \tag{65}$$

- The output PSD is then

$$\Gamma_Y(e^{j\omega}) = \sigma_e^2 \left(2H(e^{j\omega})H(e^{-j\omega}) + 3\right) \tag{66}$$

Example 11.3 - Solution

- In the case of the structure shown in Figure 17b, the transfer functions from the multiplier outputs to the filter output can be readily computed from the results of Exercise 4.3, considering that for $-\mathrm{m}_1$

$$\mathbf{b} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \quad \mathrm{d} = 0 \tag{67}$$

and for $-\mathrm{m}_2$

$$\mathbf{b} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}; \quad \mathrm{d} = 0 \tag{68}$$

Example 11.3 - Solution

- The required transfer functions are given by

$$G_{m_1}(z) = \frac{z+1}{D(z)} \tag{69}$$

$$G_{m_2}(z) = \frac{-(z-1)}{D(z)} \tag{70}$$

where

$$D(z) = z^2 + (m_1 - m_2)z + (m_1 + m_2 - 1) \tag{71}$$

- The PSD at the output is then given by

$$\Gamma_Y(e^{j\omega}) = \sigma_e^2 \left(G_{m_1}(e^{j\omega}) G_{m_1}(e^{-j\omega}) + G_{m_2}(e^{j\omega}) G_{m_2}(e^{-j\omega}) \right) \tag{72}$$

Product quantization

- In floating-point arithmetic, quantization errors are introduced not only in products but also in additions.
- In fact, the sum and product of two numbers x_1 and x_2, in finite precision, have the following characteristics

$$Fl\{x_1 + x_2\} = x_1 + x_2 - (x_1 + x_2)n_a \tag{73}$$

$$Fl\{x_1 x_2\} = x_1 x_2 - (x_1 x_2)n_p \tag{74}$$

where n_a and n_p are random variables with zero mean, which are independent of any other internal signals and errors in the filter.

Product quantization

- Their variances are approximately given by

$$\sigma^2_{n_a} \approx 0.165 \times 2^{-2b} \tag{75}$$

$$\sigma^2_{n_p} \approx 0.180 \times 2^{-2b} \tag{76}$$

  respectively, where b is the number of bits in the mantissa representation, not including the sign bit.

- Using the above expressions, the roundoff-noise analysis for floating-point arithmetic can be done in the same way as for the fixed-point case.

- A variant of floating-point arithmetic is the so-called block-floating-point arithmetic, which consists of representing several numbers with a shared exponent term.

- Block floating point is a compromise between the high complexity hardware of floating-point arithmetic and the short dynamic range inherent to the fixed-point representation.

Signal scaling

- It is possible for overflow to occur in any internal node of a digital filter structure.
- In general, input scaling is often required to reduce the probability of overflow occurring to an acceptable level.
- Particularly in fixed-point implementations, signal scaling should be applied to make the probability of overflow the same at all internal nodes of a digital filter.
- In such cases, the signal-to-noise ratio is maximized.
- In a practical digital filter, however, a few internal signals are critical, and they should not exceed the allowed dynamic range for more than some very short periods of time.

Signal scaling

- For these signals, if overflow frequently occurs, serious signal distortions will be observed at the filter output.
- If either the one's- or the two's-complement representation is used, an important fact greatly simplifies the scaling in order to avoid overflow distortions: if the sum of two or more numbers is within the available range of representable numbers, the complete sum will always be correct irrespective of the order in which they are added, even if overflow occurs in a partial operation.
- This implies that the overflow distortions which are due to signal additions can be recovered by subsequent additions.

Signal scaling

- As a consequence, when using either one's- or two's-complement arithmetic, one only needs to avoid overflow at the multiplier inputs.
- Therefore, they are the only points that require scaling.
- In this case, in order to avoid frequent overflows, we should calculate the upper limit for the magnitude of each signal $x_i(n)$, for all possible types of filter inputs $u(n)$.
- This is shown by the analysis of the decimation-in-time FFT algorithm in the example below.

Example 11.4

- Perform a roundoff-noise analysis of FFT algorithms.

Example 11.4 - Solution

- Each distinct FFT algorithm requires a specific analysis of the corresponding quantization effects.
- In this example, we perform a roundoff-noise analysis on the radix-2 FFT algorithm with decimation in time.
- The basic cell of the radix-2 FFT algorithm based on the decimation-in-time method is shown in Figure 18.

Example 11.4 - Solution

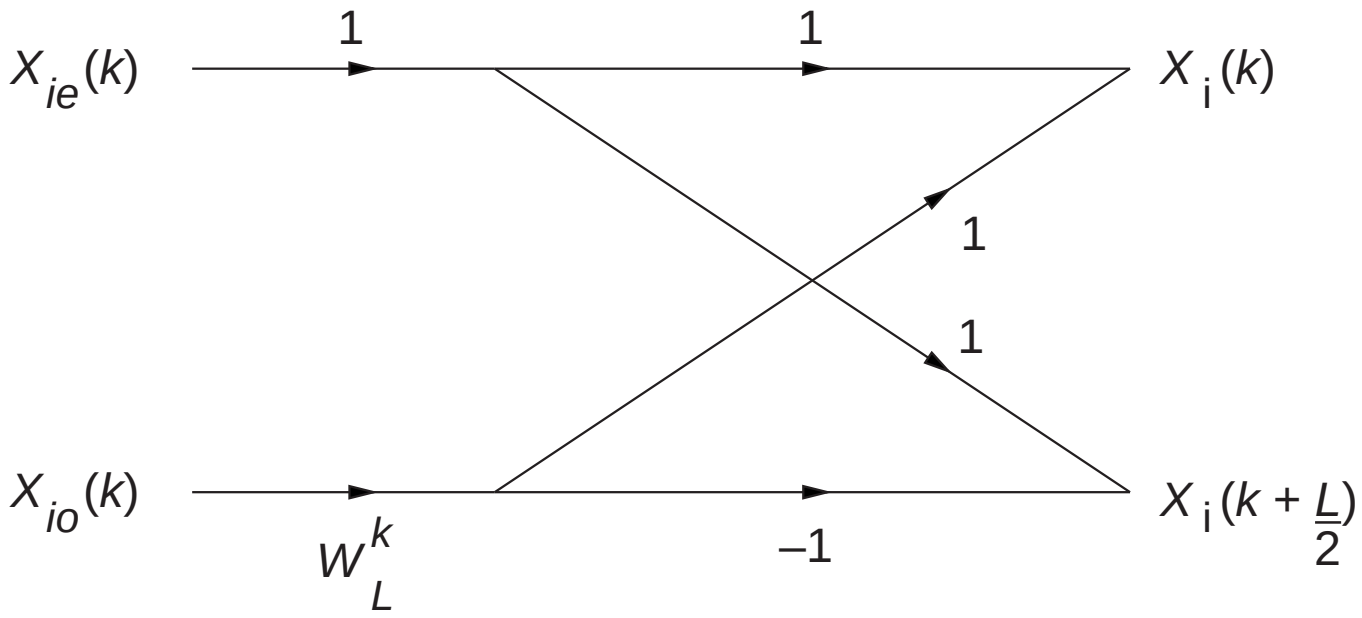


Figure 18: Basic cell of the radix-2 FFT algorithm using decimation in time.

Example 11.4 - Solution

- From Figure 18, we have that

$$|X_i(k)| \leq 2 \max\{|X_{ie}(k)|, |X_{io}(k)|\} \tag{77}$$

$$\left|X_i\left(k+\frac{L}{2}\right)\right| \leq 2 \max\{|X_{ie}(k)|, |X_{io}(k)|\} \tag{78}$$

- Therefore, to avoid overflow in such structure, using fixed-point arithmetic, a factor of $\frac{1}{2}$ on each cell input should be employed, as seen in Figure 19.
- There, one can clearly discern the two noise sources that result from both signal scaling and multiplication by W_L^k.

Example 11.4 - Solution

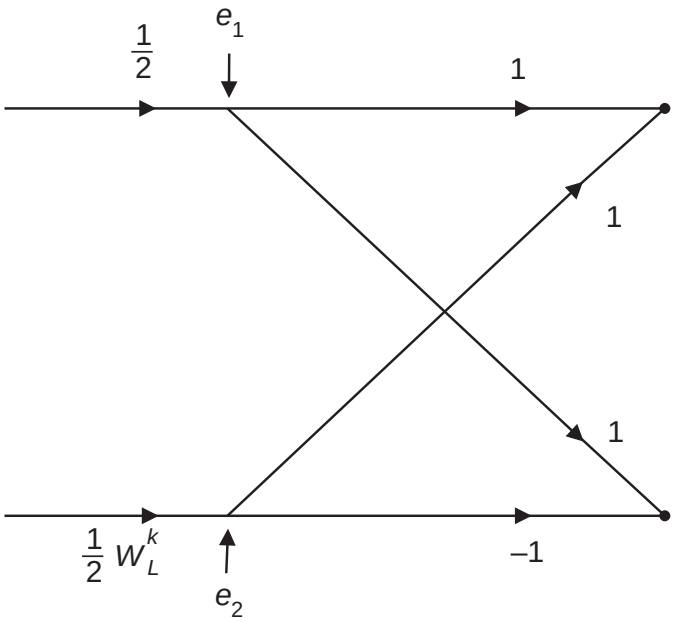


Figure 19: Basic cell with input scaling.

Example 11.4 - Solution

- In the common case of rounding, the noise variance, as given in equation (54), is equal to

$$\sigma_e^2 = \frac{2^{-2b}}{12} \tag{79}$$

  where b is the number of fixed-point bits.

- In Figure 19, the noise source e_1 models the scaling noise on a cell input which is a complex number.

- By considering the real and imaginary noise contributions to be uncorrelated, we have that

$$\sigma_{e_1}^2 = 2\sigma_e^2 = \frac{2^{-2b}}{6} \tag{80}$$

- Meanwhile, the noise source, e_2, models the scaling noise due to the multiplication of two complex numbers, which involves four different terms.

Example 11.4 - Solution

- By considering these four terms to be uncorrelated, we have that

$$\sigma_{e_2}^2 = 4\sigma_e^2 = \frac{2^{-2b}}{3} \tag{81}$$

  This completes the noise analysis in a single basic cell.

- To extend these results to the overall FFT algorithm, let us assume that all noise sources in the algorithm are uncorrelated.

- Therefore, the output-noise variance can be determined by the addition of all individual noise variances, from all the basic cells involved.

- Naturally, the noise generated in the first stage appears at the output scaled by $(\frac{1}{2})^{(l-1)}$, where l is the overall number of stages, and the noise generated at stage k is multiplied by $(\frac{1}{2})^{(l-k)}$.

Example 11.4 - Solution

- To determine the total number of cells, we note that each FFT output is directly connected to one basic cell in the last stage, two in the next-to-last stage, and so on, up to $2^{(l-1)}$ cells in the first stage, with each cell presenting two noise sources similar to e_1 and e_2, as discussed above.
- Each stage then has $2^{(l-k)}$ cells, and their output-noise contribution is given by

$$2^{(l-k)} \left(\frac{1}{2}\right)^{(2l-2k)} \left(\sigma_{e_1}^2 + \sigma_{e_2}^2\right) \tag{82}$$

- Consequently, the overall noise variance in each FFT output sample is given by

$$\sigma_o^2 = \left(\sigma_{e_1}^2 + \sigma_{e_2}^2\right) \sum_{k=1}^{l} 2^{l-k} \left(\frac{1}{2}\right)^{2l-2k} = 6\sigma_e^2 \sum_{k=1}^{l} \left(\frac{1}{2}\right)^{l-k} = 6\sigma_e^2 \left(2 - \frac{1}{2^{l-1}}\right) \tag{83}$$

- A similar analysis for other FFT algorithms can be determined in an analogous way.

Signal scaling

- The general case of scaling analysis is illustrated in Figure 20, where $F_i(z)$ and $F'_i(z)$ represent the transfer functions before and after scaling, respectively, from the input node to the input of the multiplier m_i, such that

$$F'_i(z) = \lambda F_i(z) \tag{84}$$

  which also holds for the corresponding impulse responses, that is

$$f'_i(n) = \lambda f_i(n) \tag{85}$$

  for all n.

- Assuming zero initial conditions, we have that

$$x_i(n) = \sum_{k=0}^{\infty} f'_i(k)u(n-k) = \lambda \sum_{k=0}^{\infty} f_i(k)u(n-k) \tag{86}$$

Signal scaling

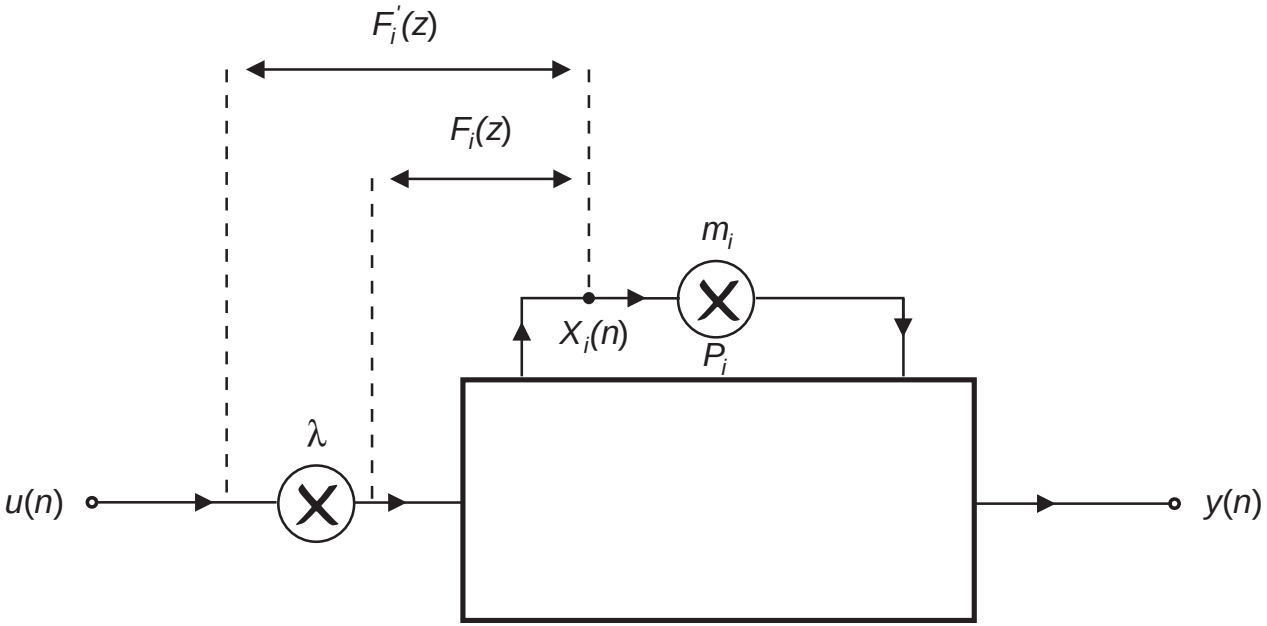


Figure 20: Signal scaling.

Signal scaling

- If $u(n)$ is bounded in magnitude by u_m, for all n, the above equation implies that

$$|x_i(n)| \leq u_m \sum_{k=0}^{\infty} |f_i'(k)| = u_m \lambda \sum_{k=0}^{\infty} |f_i(k)| \tag{87}$$

- If we want the magnitude of the signal $x_i(n)$ to also be upper bounded by u_m, for all types of input sequences, then the associated scaling must ensure that

$$\sum_{k=0}^{\infty} |f_i'(k)| \leq 1 \tag{88}$$

and therefore

$$\lambda \leq \frac{1}{\sum_{k=0}^{\infty} |f_i(k)|} \tag{89}$$

- This is a necessary and sufficient condition to avoid overflow for any input signal.

Signal scaling

- However, the condition in equations (88) and (89) is not useful in practice, as it can not be easily implemented. In addition, for a large class of input signals, it leads to a very stringent scaling.
- A more practical scaling strategy, aimed at more specific classes of input signals, is presented in the following. Since

$$U(z) = \sum_{n=-\infty}^{\infty} u(n) z^{-n} \tag{90}$$

and

$$X_i(z) = F_i'(z) U(z) = \lambda F_i(z) U(z) \tag{91}$$

then, in the time domain, $x_i(n)$ is given by

$$x_i(n) = \frac{1}{2\pi j} \oint_C X_i(z) z^{n-1} dz \tag{92}$$

where C is in the convergence region common to $F_i(z)$ and $U(z)$.

Signal scaling

- Accordingly, in the frequency domain, $x_i(n)$ is given by

$$x_i(n) = \frac{\lambda}{2\pi} \int_0^{2\pi} F_i(e^{j\omega}) U(e^{j\omega}) e^{j\omega n} d\omega \tag{93}$$

- Let, now, the L_p norm be defined for any periodic function $F(e^{j\omega})$ as follows:

$$\|F(e^{j\omega})\|_p = \left(\frac{1}{2\pi} \int_0^{2\pi} |F(e^{j\omega})|^p d\omega \right)^{\frac{1}{p}} \tag{94}$$

for all $p \geq 1$, such that $\int_0^{2\pi} |F(e^{j\omega})|^p d\omega \leq \infty$.

- If $F(e^{j\omega})$ is continuous, the limit when $p \to \infty$ of equation (94) exists, and is given by

$$\|F(e^{j\omega})\|_\infty = \max_{0 \leq \omega \leq 2\pi} \left\{ |F(e^{j\omega})| \right\} \tag{95}$$

Signal scaling

- Assuming that $|U(e^{j\omega})|$ is upper bounded by U_m, that is, $\|U(e^{j\omega})\|_\infty \leq U_m$, it clearly follows from equation (93) that

$$|x_i(n)| \leq \frac{U_m \lambda}{2\pi} \int_0^{2\pi} |F_i(e^{j\omega})| d\omega \tag{96}$$

that is

$$|x_i(n)| \leq \lambda \|F_i(e^{j\omega})\|_1 \|U(e^{j\omega})\|_\infty \tag{97}$$

- Following a similar reasoning, we have that

$$|x_i(n)| \leq \lambda \|F_i(e^{j\omega})\|_\infty \|U(e^{j\omega})\|_1 \tag{98}$$

Signal scaling

- Also, from the Schwartz inequality

$$|x_i(n)| \leq \lambda \|F_i(e^{j\omega})\|_2 \|U(e^{j\omega})\|_2 \tag{99}$$

- Equations (97)–(99) are special cases of a more general relation, known as the Hölder inequality, which states that, if $\frac{1}{p} + \frac{1}{q} = 1$, then

$$|x_i(n)| \leq \lambda \|F_i(e^{j\omega})\|_p \|U(e^{j\omega})\|_q \tag{100}$$

- If $|u(n)| \leq u_m$, for all n, and if its Fourier transform exists, then there is U_m such that $\|U(e^{j\omega})\|_q \leq U_m$, for any $q \geq 1$.

- If we then want $|x_i(n)|$ to be upper limited by U_m, for all n, equation (100) indicates that a proper scaling factor should be such that

$$\lambda \leq \frac{1}{\|F_i(e^{j\omega})\|_p} \tag{101}$$

Signal scaling

- In practice, when the input signal is deterministic, the most common procedures for determining λ are:
  - When $U(e^{j\omega})$ is bounded, and therefore $\|U(e^{j\omega})\|_\infty$ can be precisely determined, one may use λ as in equation (101), with $p = 1$.
  - When the input signal has finite energy, that is,

$$E = \sum_{n=-\infty}^{\infty} u^2(n) = \|U(e^{j\omega})\|_2^2 < \infty \qquad (102)$$

    then λ can be obtained from equation (101) with $p = 2$.
  - If the input signal has a dominant frequency component, such as a sinusoidal signal, this means that it has an impulse in the frequency domain. In this case, neither $\|U(e^{j\omega})\|_\infty$ nor $\|U(e^{j\omega})\|_2$ are defined, and thus only the L_1 norm can be used. Then the scaling factor comes from equation (101) with $p = \infty$, which is the most strict case for λ.

Signal scaling

- For the random-input case, the above analysis does not apply directly, since the z transform of $u(n)$ is not defined. In this case, if $u(n)$ is stationary, the PSD of an internal signal $x_i(n)$ is given by

$$\Gamma_{x_i}(e^{j\omega}) = F_i'(e^{j\omega})F_i'(e^{-j\omega})\Gamma_u(e^{j\omega}) = \lambda^2 F_i(e^{j\omega})F_i(e^{-j\omega})\Gamma_u(e^{j\omega}) \quad (103)$$

where $\Gamma_u(e^{j\omega})$ is the input signal PSD.

Signal scaling

- Hence, the variance of the internal signal $x_i(n)$ is given by

$$\begin{aligned}
\sigma_{x_i}^2 &= R_{X_i}(0) \\
&= \frac{1}{2\pi}\int_0^{2\pi} \Gamma_{X_i}(e^{j\omega})e^{j\omega\nu}\,d\omega\Big|_{\nu=0} \\
&= \frac{1}{2\pi}\int_0^{2\pi} \Gamma_{X_i}(e^{j\omega})\,d\omega \\
&= \frac{1}{2\pi}\int_0^{2\pi} |F_i'(e^{j\omega})|^2\Gamma_U(e^{j\omega})\,d\omega \\
&= \frac{\lambda^2}{2\pi}\int_0^{2\pi} |F_i(e^{j\omega})|^2\Gamma_U(e^{j\omega})\,d\omega \qquad (104)
\end{aligned}$$

Signal scaling

- Applying the Hölder inequality (equation (100)) to the above equation, we have that, if $\frac{1}{p} + \frac{1}{q} = 1$, then

$$\sigma_{x_i}^2 \leq \lambda^2 \|F_i^2(e^{j\omega})\|_p \|\Gamma_U(e^{j\omega})\|_q \tag{105}$$

or, alternatively,

$$\sigma_{x_i}^2 \leq \lambda^2 \|F_i(e^{j\omega})\|_{2p}^2 \|\Gamma_U(e^{j\omega})\|_q \tag{106}$$

Signal scaling

- For random processes, the most interesting cases in practice are:
  - If we consider $q = 1$, then $p = \infty$, and noting that $\sigma_u^2 = \|\Gamma_U(e^{j\omega})\|_1$, we have, from equation (106), that

$$\sigma_{x_i}^2 \leq \lambda^2 \|F_i(e^{j\omega})\|_\infty^2 \, \sigma_u^2 \tag{107}$$

  and a λ, such that $\sigma_{x_i}^2 \leq \sigma_u^2$, is

$$\lambda = \frac{1}{\|F_i(e^{j\omega})\|_\infty} \tag{108}$$

Signal scaling

- (cont.)
  - If the input signal is a white noise, $\Gamma_U(e^{j\omega}) = \sigma_u^2$, for all ω, and then, from equation (103),

$$\sigma_{x_i}^2 = \lambda^2 \|F_i(e^{j\omega})\|_2^2 \sigma_u^2 \tag{109}$$

and an appropriate λ is

$$\lambda = \frac{1}{\|F_i(e^{j\omega})\|_2} \tag{110}$$

Signal scaling

- In practical implementations, the use of powers of two to represent the scaling multiplier coefficients is a common procedure, as long as these coefficients satisfy the constraints to control overflow.
- In this manner, the scaling multipliers can be implemented using simple shift operations.
- In the general case when we have m multipliers, the following single scaling can be used at the input:

$$\lambda = \frac{1}{\max\{\|F_1(e^{j\omega})\|_p, \|F_2(e^{j\omega})\|_p, \ldots, \|F_m(e^{j\omega})\|_p\}} \tag{111}$$

Signal scaling

- For cascade and parallel realizations, a scaling multiplier is employed at the input of each section.
- For some types of second-order sections, used in cascade realizations, the scaling factor of a given section can be incorporated into the output multipliers of the previous section. general, this procedure leads to a reduction in the output quantization noise.
- In the case of second-order sections, it is possible to calculate the L_2 and L_∞ norms of the internal transfer functions in closed form, as given in Chapter 13.

Example 11.5

- Scale the filter shown in Figure 21 using the L_2 norm, aiming at a possible implementation in a fixed-point machine, and determine the relative noise variance at the scaled filter output.

Example 11.5

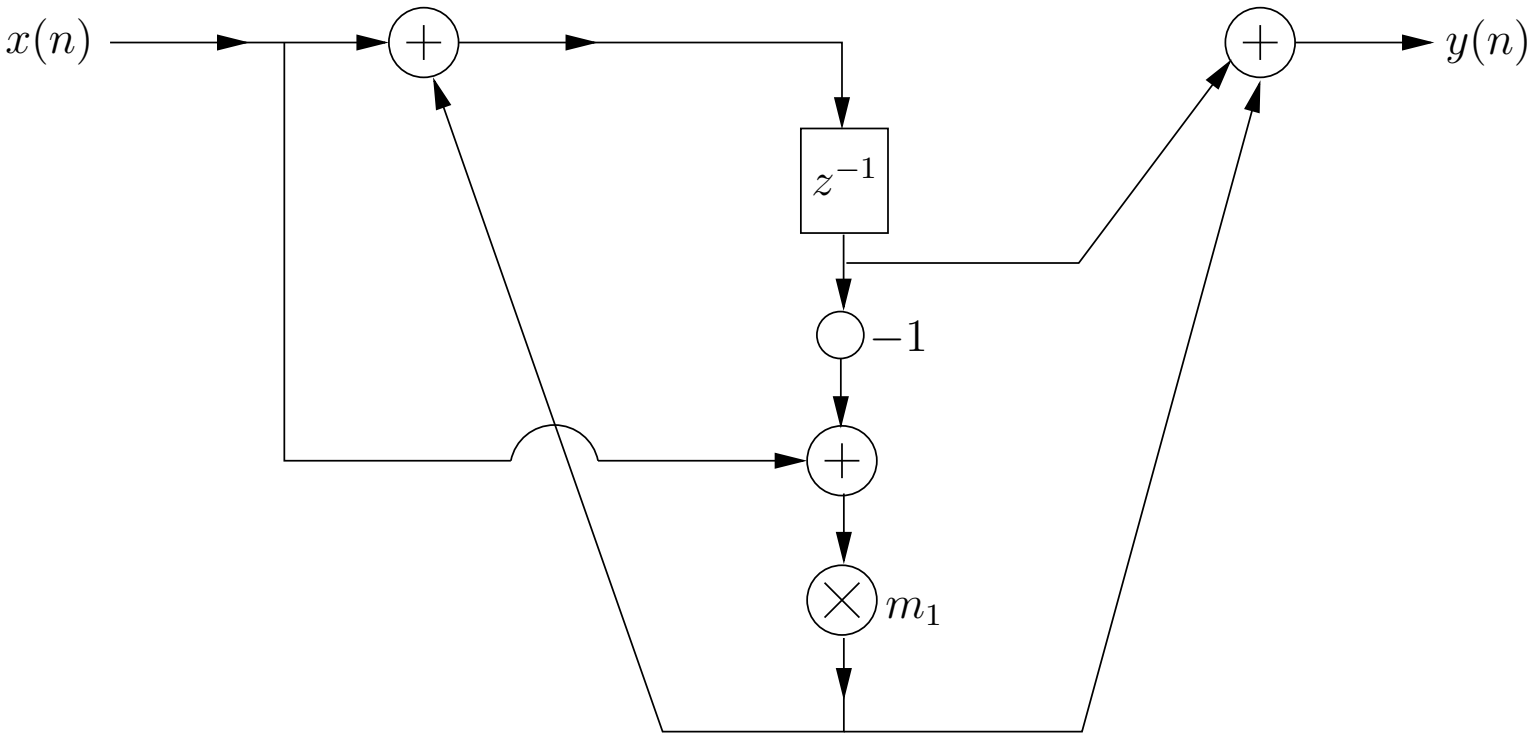


Figure 21: Filter structure in Example 11.5.

Example 11.5 - Solution

- Denoting the delay input as $s(n+1)$, one can easily infer that

$$\left.\begin{aligned} s(n+1) &= x(n) + m_1(x(n) - s(n)) \\ y(n) &= s(n) + m_1(x(n) - s(n)) \end{aligned}\right\} \tag{112}$$

or equivalently

$$\left.\begin{aligned} s(n+1) &= -m_1 s(n) + (1+m_1)x(n) \\ y(n) &= (1-m_1)s(n) + m_1 x(n) \end{aligned}\right\} \tag{113}$$

which corresponds to the state-space description characterized by

$$\mathbf{A} = -m_1;\ \ \mathbf{B} = (1+m_1);\ \ \mathbf{C} = (1-m_1);\ \ d = m_1 \tag{114}$$

Example 11.5 - Solution

- The transfer function for this example is then

$$H(z) = \frac{(1-m_1)(1+m_1)}{(z+m_1)} + m_1 = \frac{m_1 z + 1}{z + m_1} \tag{115}$$

which happens to be an all-pass first-order transfer function such that

$$\|H(z)\|_2^2 = 1 \tag{116}$$

- For scaling, it is required the computation of the transfer function from the filter input to the input of the m_1 multiplier, obtained, in this case, by setting $\mathbf{C} = -1$ and $d = 1$, such that

$$F_1(z) = -\frac{1+m_1}{z+m_1} + 1 = \frac{z-1}{z+m_1} \tag{117}$$

Example 11.5 - Solution

- In order to determine the scaling factor, we need to compute

$$\|F_1(z)\|_2 = \sqrt{\frac{1}{2\pi}\int_0^{2\pi} |F_1(e^{j\omega})|^2 d\omega} = \sqrt{\frac{1}{2\pi j}\oint_C F_1(z)F_1(z^{-1})z^{-1}dz} \quad (118)$$

  where the integral contour C is the z-domain unit circle.

- Therefore, using the residue theorem, one can write that

$$\|F_1(z)\|_2^2 = \sum_{\text{residues}} \left[F_1(z)F_1(z^{-1})z^{-1}\right] \quad (119)$$

  where the residues are determined for all poles of $F_1(z)F_1(z^{-1})z^{-1}$ within C.

Example 11.5 - Solution

- In this example, assuming $|m_1| < 1$, we get

$$\begin{aligned} \|F_1(z)\|_2^2 &= \sum_{\text{residues}} \frac{(z-1)(1-z)}{(z+m_1)(1+zm_1)z} \\ &= \frac{(-m_1-1)(1+m_1)}{(1-m_1^2)(-m_1)} - \frac{1}{m_1} \\ &= \frac{2}{1-m_1} \end{aligned} \tag{120}$$

leading to a scaling factor of

$$\lambda = \sqrt{\frac{1-m_1}{2}} \tag{121}$$

which shall be compensated in the filter output by a gain $g = \sqrt{\frac{2}{1-m_1}}$.

Example 11.5 - Solution

- The calculation of the output noise variance requires the transfer function from the multiplier output to the filter output, which, in this case, can be obtained by substituting $\mathbf{B} = d = 1$ in the state-space representation, such that

$$G_1(z) = \frac{1 - m_1}{z + m_1} + 1 = \frac{z + 1}{z + m_1} \tag{122}$$

- Again, by employing the residue theorem, we get

$$\begin{aligned} \|G_1(z)\|_2^2 &= \sum_{\text{residues}} \left[G_1(z) G_1(z^{-1}) z^{-1} \right] \\ &= \sum_{\text{residues}} \frac{(1+z)^2}{(z + m_1)(1 + z m_1) z} \\ &= \frac{(1 - m_1)^2}{(1 - m_1^2)(-m_1)} + \frac{1}{m_1} \\ &= \frac{2}{1 + m_1} \end{aligned} \tag{123}$$

- Hence,

$$\begin{aligned}\frac{\sigma_y^2}{\sigma_e^2} &= \|H(z)\|_2^2 g^2 + \|G_1(z)\|_2^2 g^2 + 1 \\ &= \frac{2}{1-m_1} + \frac{2}{1+m_1}\frac{2}{1-m_1} + 1 \\ &= \frac{-m_1^2 + 2m_1 + 7}{m_1^2 - 1}\end{aligned} \tag{124}$$

already taking into account the scaling multiplier at the filter input and the compensating gain at the filter output.

Coefficient quantization

- During the approximation step, the coefficients of a digital filter are calculated with the high accuracy inherent to the computer employed in the design.
- When these coefficients are quantized for practical implementations, commonly using rounding, the time and frequency responses of the realized digital filter deviate from the ideal response.
- In fact, the quantized filter may even fail to meet the prescribed specifications.
- The sensitivity of the filter response to errors in the coefficients is highly dependent on the type of structure.
- This fact is one of the motivations for considering alternative realizations having low sensitivity, such as those presented in Chapter 13.

Coefficient quantization

- Among the several sensitivity criteria that evaluate the effect of the fixed-point coefficient quantization on the digital filter transfer function, the most widely used are

$$ {}_{\mathrm{I}}S_{m_i}^{H(z)}(z) = \frac{\partial H(z)}{\partial m_i} \tag{125} $$

$$ {}_{\mathrm{II}}S_{m_i}^{H(z)}(z) = \frac{1}{H(z)} \frac{\partial H(z)}{\partial m_i} \tag{126} $$

- For the floating-point representation, the sensitivity criterion must take into account the relative variation of $H(z)$ due to a relative variation in a multiplier coefficient. We then must use

$$ {}_{\mathrm{III}}S_{m_i}^{H(z)}(z) = \frac{m_i}{H(z)} \frac{\partial H(z)}{\partial m_i} \tag{127} $$

Coefficient quantization

- With such a formulation, it is possible to use the value of the multiplier coefficient to determine ${}_{\mathrm{III}}S_{m}^{H(z)}(z)$.
- A simple example illustrating the importance of this fact is given by the quantization of the system

$$y(n) = (1+m)x(n); \text{ for } |m| \ll 1 \tag{128}$$

- Using equation (125), ${}_{\mathrm{I}}S_{m_i}^{H(z)}(z) = 1$, regardless of the value of m, while using equation (127), ${}_{\mathrm{III}}S_{m_i}^{H(z)}(z) = m/(m+1)$, indicating that a smaller value for the magnitude of m leads to a smaller sensitivity of $H(z)$ with respect to m.
- This is true for the floating-point representation, as long as the number of bits in the exponent is enough to represent the exponent of m.

Example 11.6

- Determine the possible pole positions for a direct-form second-order section with denominator

$$D(z) = z^2 + a_1 z + a_2 \tag{129}$$

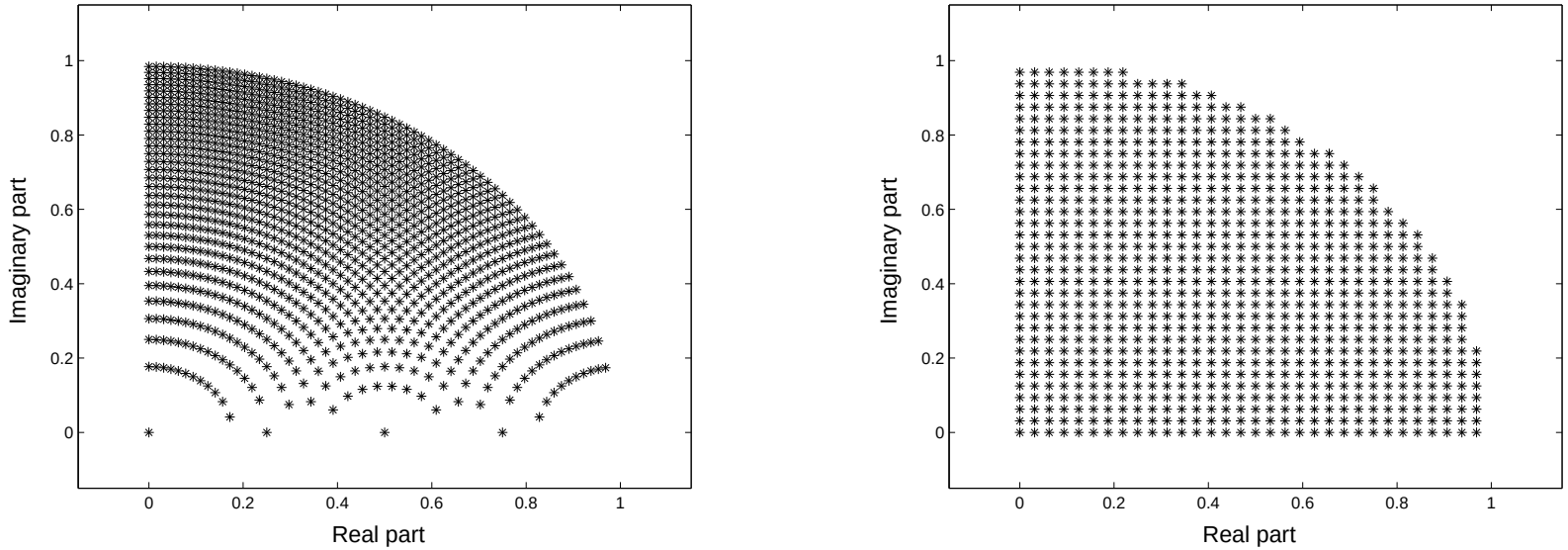
when the filter coefficients a_1 and a_2 are represented with 6 bits, including the sign bit, using standard binary representation.

- Repeat your analysis using a state-space structure characterized by $a_{11} = a_{22} = a$ and $a_{21} = -a_{12} = \zeta$, when a and ζ are represented with 6 bits. Such a structure, when the values of at least two different coefficients are dependent on a single parameter, is referred to as a coupled-form state-space structure.

Example 11.6 - Solution

- Figure 22a depicts the possible pole placements in the first quadrant within the z-domain unit circle for the direct-form second-order section.
- In the remaining quadrants, pole placements are symmetric mirrored copies of the ones seen in this figure.
- As can be observed, the pole grid becomes very sparse around to the real axis, particularly close to $z = 0$ or $z = 1$.
- This explains the implementation inaccuracy achieved by this section type in applications with high sampling rate, since these cases often require a filter with poles close to the real axis.
- The same phenomenon occurs if the poles are required to be close to $z = 1$ or $z = -1$.

Example 11.6 - Solution



(a) (b)

Figure 22: Pole grid for second-order sections with 6-bit coefficients: (a) direct-form; (b) state-space coupled form.

Example 11.6 - Solution

- For the coupled form, the denominator polynomial becomes

$$D(z) = z^2 - 2\alpha z + \alpha^2 + \zeta^2 \tag{130}$$

  where α represents the real part of the complex conjugate poles and ζ their imaginary part.

- The pole-placement analysis for the coefficient quantization in this structure is shown in Figure 22b, where we observe a uniform grid distribution around the entire quadrant.

- This result implies that there is no preferred region for this structure to place the poles, which is an attractive feature obtained at the cost of four multiplier coefficients to position a single pair of complex conjugate poles.

Deterministic sensitivity criterion

- In practice, one is often interested in the variation of the magnitude of the transfer function, $|H(e^{j\omega})|$, with coefficient quantization. Taking into account the contributions of all multipliers, a useful figure of merit related to this variation would be

$$S(e^{j\omega}) = \sum_{i=1}^{K} \left| S_{m_i}^{|H(e^{j\omega})|}(e^{j\omega}) \right| \tag{131}$$

  where K is the total number of multipliers in the structure, and $S_{m_i}^{|H(e^{j\omega})|}(e^{j\omega})$ is computed according to one of the equations (125)–(127), depending on the case.

- However, in general, the sensitivities of $H(e^{j\omega})$ to coefficient quantization are much easier to derive than the ones of $|H(e^{j\omega})|$, and thus it would be convenient if the first one could be used as an estimate of the second.

Deterministic sensitivity criterion

- In order to investigate this possibility, we write the frequency response in terms of its magnitude and phase as

$$H(e^{j\omega}) = \left|H(e^{j\omega})\right| e^{j\Theta(\omega)} \tag{132}$$

- Then the sensitivity measures, defined in equations (125)–(127), can be written as

$$\left|{}_{I}S_{m_i}^{H(e^{j\omega})}(e^{j\omega})\right| = \sqrt{\left({}_{I}S_{m_i}^{|H(e^{j\omega})|}(e^{j\omega})\right)^2 + |H(e^{j\omega})|^2 \left(\frac{\partial\Theta(\omega)}{\partial m_i}\right)^2} \tag{133}$$

$$\left|{}_{II}S_{m_i}^{H(e^{j\omega})}(e^{j\omega})\right| = \sqrt{\left({}_{II}S_{m_i}^{|H(e^{j\omega})|}(e^{j\omega})\right)^2 + \left(\frac{\partial\Theta(\omega)}{\partial m_i}\right)^2} \tag{134}$$

$$\left|{}_{III}S_{m_i}^{H(e^{j\omega})}(e^{j\omega})\right| = \sqrt{\left({}_{III}S_{m_i}^{|H(e^{j\omega})|}(e^{j\omega})\right)^2 + |m_i|^2 \left(\frac{\partial\Theta(\omega)}{\partial m_i}\right)^2} \tag{135}$$

- From equations (133)–(135), one can see that $|S_{m_i}^{H(e^{j\omega})}(e^{j\omega})| \geq |S_{m_i}^{|H(e^{j\omega})|}(e^{j\omega})|$.

Deterministic sensitivity criterion

- Thus, $|S_{m_i}^{H(e^{j\omega})}(e^{j\omega})|$ can be used as a conservative estimate of $|S_{m_i}^{|H(e^{j\omega})|}(e^{j\omega})|$, in the sense that it guarantees that the transfer function variation will be below a specified tolerance.

- Moreover, it is known that low sensitivity is more critical when implementing filters with poles close to the unit circle, and, in such cases, for ω close to a pole frequency ω_0, we can show that $|S_{m_i}^{H(e^{j\omega})}(e^{j\omega})| \approx |S_{m_i}^{|H(e^{j\omega})|}(e^{j\omega})|$.

- Therefore, we can rewrite equation (131), yielding the following practical sensitivity figure of merit:

$$S(e^{j\omega}) = \sum_{i=1}^{K} \left| S_{m_i}^{H(e^{j\omega})}(e^{j\omega}) \right| \tag{136}$$

where, depending on the case, $S_{m_i}^{H(e^{j\omega})}(e^{j\omega})$ is given by one of the equations (125)–(127).

Example 11.7

- Design a lowpass elliptic filter with the following specifications:

$$\left.\begin{aligned} A_p &= 1.0 \text{ dB} \\ A_r &= 40 \text{ dB} \\ \omega_p &= 0.3\pi \text{ rad/sample} \\ \omega_r &= 0.4\pi \text{ rad/sample} \end{aligned}\right\} \tag{137}$$

- Perform the fixed-point sensitivity analysis for the direct-order structure, determining the variation on the ideal magnitude response for an 11-bit quantization of the fractional part, including the sign bit.

Example 11.7 - Solution

- The coefficients of the lowpass elliptic filter are given in Table 1.

Table 1: Filter coefficients for the specifications (137).

Numerator coefficients		Denominator coefficients	
$b_0 =$	0.028 207 76	$a_0 =$	1.000 000 00
$b_1 =$	−0.001 494 75	$a_1 =$	−3.028 484 73
$b_2 =$	0.031 747 58	$a_2 =$	4.567 772 20
$b_3 =$	0.031 747 58	$a_3 =$	−3.900 153 49
$b_4 =$	−0.001 494 75	$a_4 =$	1.896 641 38
$b_5 =$	0.028 207 76	$a_5 =$	−0.418 854 19

Example 11.7 - Solution

- For the general direct-form structure described by

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_N z^{-N}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}} \tag{138}$$

  it is easy to find that the sensitivities as defined in equation (125) with respect to the numerator and denominator coefficients are given by

$${}_I S_{b_i}^{H(z)}(z) = \frac{z^{-i}}{A(z)}; \quad {}_I S_{a_i}^{H(z)}(z) = -\frac{z^{-i} H(z)}{A(z)} \tag{139}$$

  respectively.

- The magnitude of these functions for the designed fifth-order elliptic filter are seen in Figure 23.

Example 11.7 - Solution

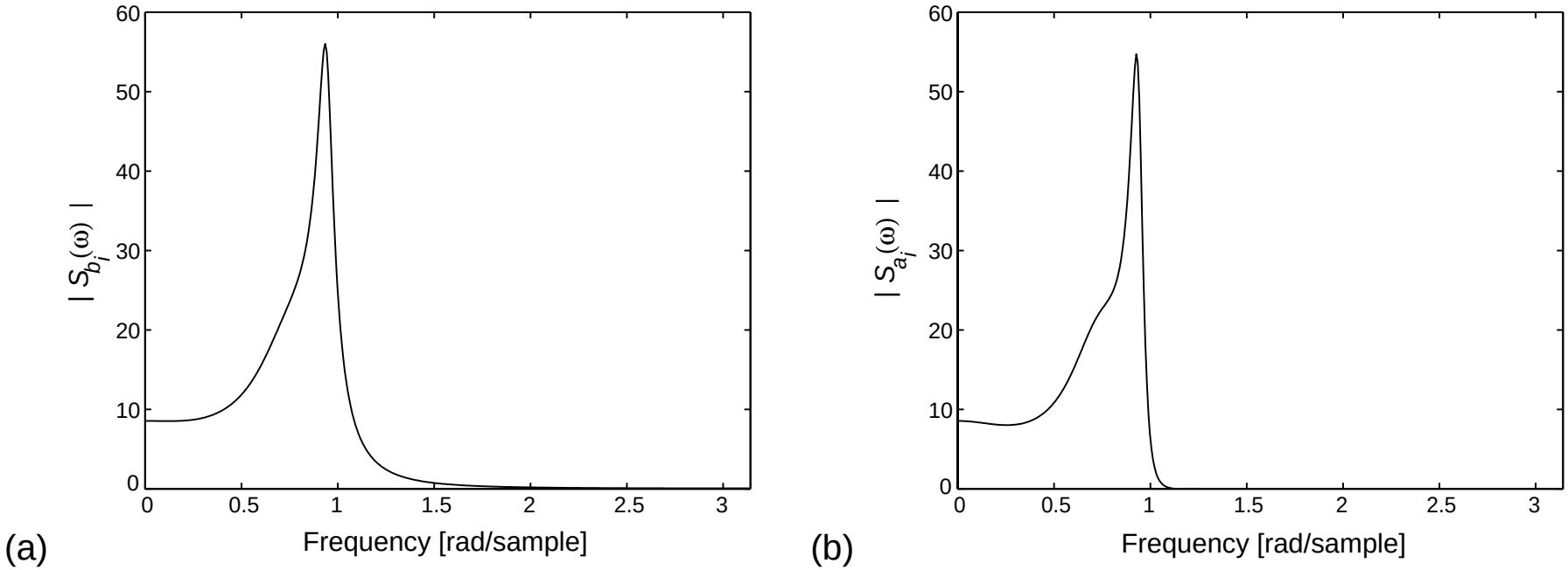


Figure 23: Magnitudes of the sensitivity functions of $H(z)$ with respect to: (a) numerator coefficients b_i; (b) denominator coefficients a_i.

Example 11.7 - Solution

- The figure of merit $S(e^{j\omega})$, as given in equation (136), for the general direct-form realization can be written as

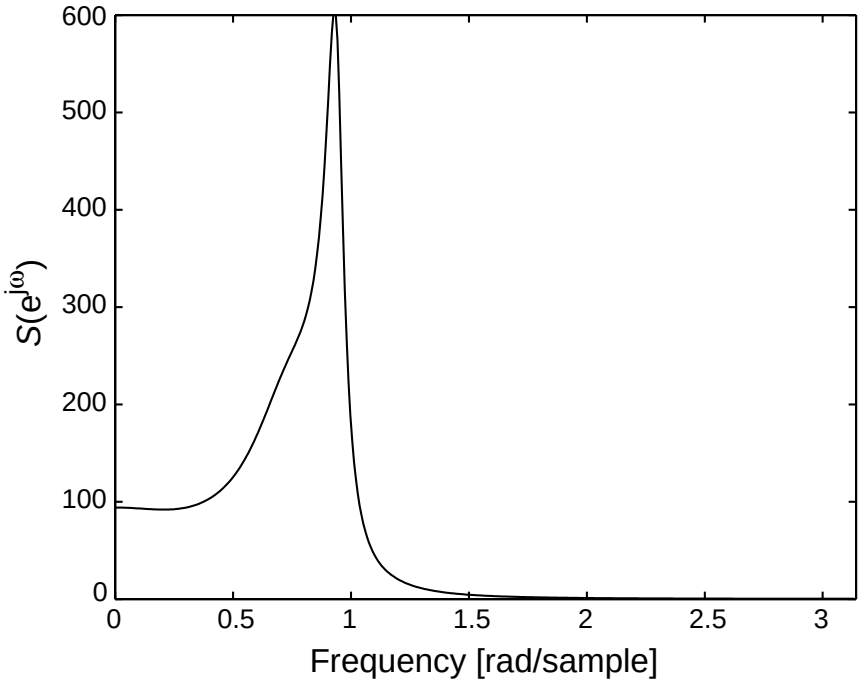
$$S(e^{j\omega}) = \frac{(N+1) + N|H(z)|}{|A(z)|} \tag{140}$$

- For this example, the function is depicted in Figure 24a.
- We can then estimate the variation of the ideal magnitude response using the approximation

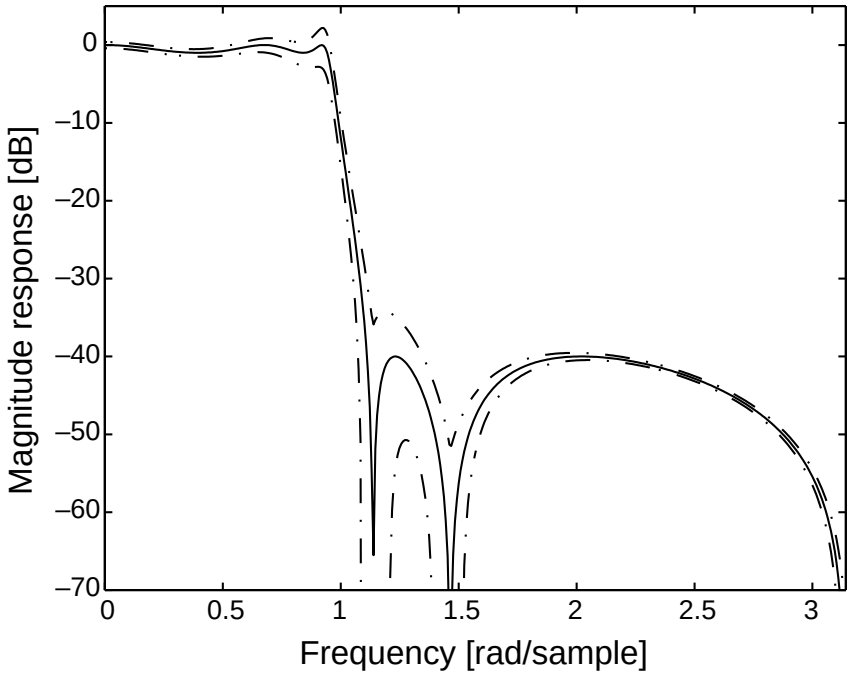
$$\Delta|H(e^{j\omega})| \approx \Delta m_i S(e^{j\omega}) \tag{141}$$

- For a fixed-point 11-bit rounding quantization, including the sign bit, $\max\{\Delta m_i\} = 2^{-11}$.
- In this case, Figure 24b depicts the ideal magnitude response of a fifth-order elliptic filter, satisfying the specifications in equation (137), along with the corresponding worst-case margins due to the coefficient quantization.

Example 11.7 - Solution



(a)



(b)

Figure 24: Finite-precision analysis: (a) sensitivity measurement $S(e^{j\omega})$; (b) worst-case variation of $|H(e^{j\omega})|$ with an 11-bit fixed-point quantization.

Deterministic sensitivity criterion

- It is worth noting that the sensitivity measurement given by equation (136) is also useful as a figure of merit if one uses the so-called pseudo-floating-point representation, that consists of implementing multiplication between a signal and a coefficient with small magnitude in the following form, as depicted in Figure 25

$$[x \times m_i]_Q = [(x \times m_i \times 2^L) \times 2^{-L}]_Q \tag{142}$$

  where L is the exponent of m_i when represented in floating point.

- Note that in the pseudo-floating-point scheme, all operations are actually performed using fixed-point arithmetic.

Deterministic sensitivity criterion

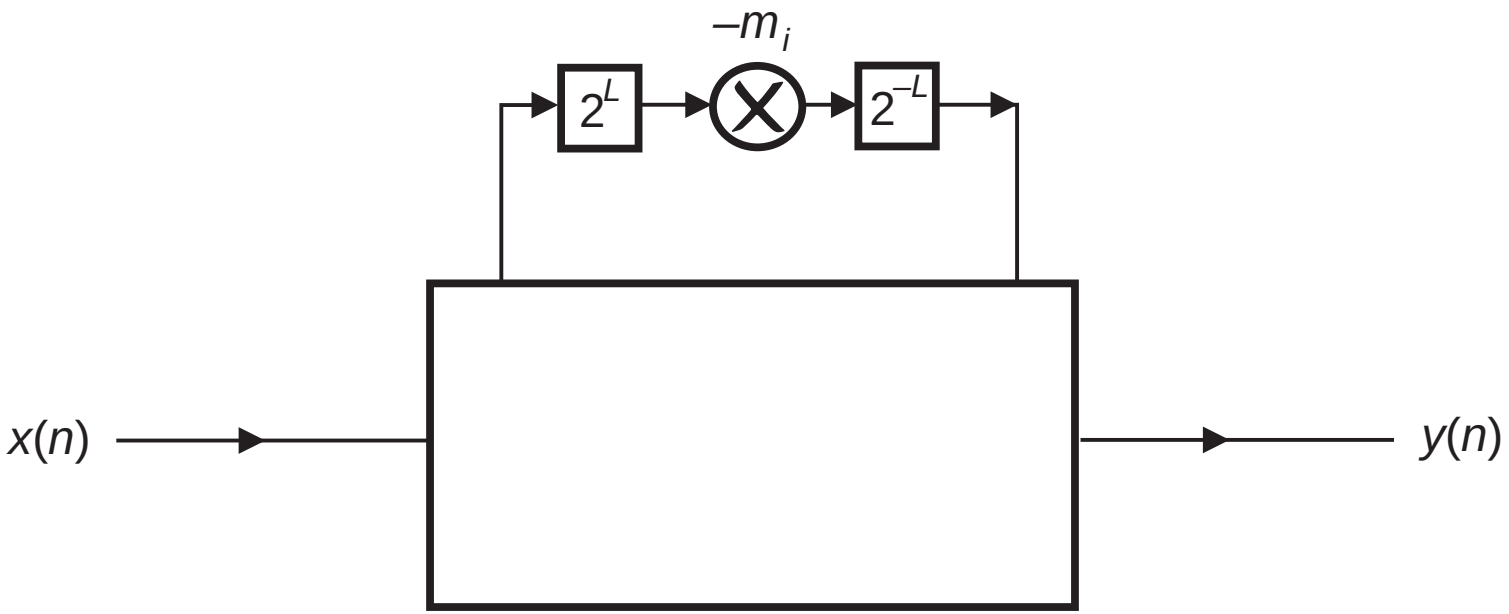


Figure 25: Implementation of a multiplication in a pseudo-floating-point representation.

Statistical forecast of the wordlength

- In the previous subsection we computed the worst case variation in the frequency response of a digital filter with the quantization of coefficients. By worst case we meant the supposition that quantization made all the coefficients to vary the maximum possible amount, and in the worst direction.
- However, since it is unlikely that all coefficients will undergo a worst-case quantization error, and their quantization effects will accumulate in the worst possible way with respect to the resulting frequency response.
- Thus, it is useful to perform a more realistic, statistical analysis of the deviation in the frequency response.

Statistical forecast of the wordlength

- In this subsection we perform a statistical forecast of the wordlength necessary for a filter to satisfy a given specification.
- Suppose we have designed a digital filter with frequency response $\mathrm{H}(\mathrm{e}^{\mathrm{j}\omega})$, and that the ideal magnitude response is $\mathrm{H_d}(\mathrm{e}^{\mathrm{j}\omega})$, with a tolerance given by $\rho(\omega)$.
- When the filter coefficients are quantized, we can express the resulting magnitude response as

$$\left|\mathrm{H_Q}(\mathrm{e}^{\mathrm{j}\omega})\right| = \left|\mathrm{H}(\mathrm{e}^{\mathrm{j}\omega})\right| + \Delta\left|\mathrm{H}(\mathrm{e}^{\mathrm{j}\omega})\right| \tag{143}$$

Statistical forecast of the wordlength

- Obviously, for a meaningful design, $|H_Q(e^{j\omega})|$ must not deviate from $H_d(e^{j\omega})$ by more than a frequency-dependent tolerance $\rho(\omega)$, that is

$$\left|\left(\left|H_Q(e^{j\omega})\right| - H_d(e^{j\omega})\right)\right| = \left|\left|H(e^{j\omega})\right| + \Delta\left|H(e^{j\omega})\right| - H_d(e^{j\omega})\right| \leq \rho(\omega) \tag{144}$$

  or, more strictly,

$$\left|\Delta\left|H(e^{j\omega})\right|\right| \leq \rho(\omega) - \left|\left|H(e^{j\omega})\right| - H_d(e^{j\omega})\right| \tag{145}$$

- The variation in the magnitude response of the digital filter due to the variations in the multiplier coefficients m_i can be approximated by

$$\Delta\left|H(e^{j\omega})\right| \approx \sum_{i=1}^{K} \frac{\partial\left|H(e^{j\omega})\right|}{\partial m_i}\Delta m_i \tag{146}$$

Statistical forecast of the wordlength

- If we consider that:
  - the multiplier coefficients are rounded;
  - the quantization errors are statistically independent;
  - all Δm_i are uniformly distributed;

  then the variance of the error in each coefficient, based on equation (54), is given by

$$\sigma^2_{\Delta m_i} = \sigma^2_{\Delta m} = \frac{2^{-2b}}{12}, \quad \text{for} \quad i = 1, 2, \ldots, K \tag{147}$$

  where b is the number of bits not including the sign bit.

Statistical forecast of the wordlength

- With the assumptions above, the mean of $\Delta|H(e^{j\omega})|$ is zero and its variance is given by

$$\sigma^2_{\Delta|H(e^{j\omega})|} \approx \sigma^2_{\Delta m} \sum_{i=1}^{K} \left(\frac{\partial |H(e^{j\omega})|}{\partial m_i} \right)^2 = \sigma^2_{\Delta m} S^2(e^{j\omega}) \tag{148}$$

  where $S^2(e^{j\omega})$ is given by equations (125) and (136).

- If we further assume that $\Delta|H(e^{j\omega})|$ is Gaussian, we can estimate the probability of $\Delta|H(e^{j\omega})|$ being less than or equal to $x\sigma_{\Delta|H(e^{j\omega})|}$ by

$$\Pr\left\{ \left| \Delta H(e^{j\omega}) \right| \leq x\sigma_{\Delta|H(e^{j\omega})|} \right\} = \frac{2}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{2}}} e^{-x'^2} dx' \tag{149}$$

Statistical forecast of the wordlength

- To guarantee that equation (145) holds with a probability less than or equal to the one given in equation (149), it suffices that

$$x\sigma_{\Delta m} S(e^{j\omega}) \leq \rho(\omega) - \left| \left| H(e^{j\omega}) \right| - H_d(e^{j\omega}) \right| \tag{150}$$

- Now, assume that the wordlength, including the sign bit, is given by

$$B = I + F + 1 \tag{151}$$

where I and F are the numbers of bits in the integer and fractional parts, respectively.

- The value of I depends on the required order of magnitude of the coefficient, and F can be estimated from equation (150) to guarantee that equation (145) holds with probability as given in equation (149).

Statistical forecast of the wordlength

- To satisfy the inequality in (150), the value of 2^{-b}, from equation (147), should be given by

$$2^{-b} = \sqrt{12} \min_{\omega \in C} \left\{ \left| \frac{\rho(\omega) - ||H(e^{j\omega})| - H_d(e^{j\omega})|}{xS(e^{j\omega})} \right| \right\} \tag{152}$$

  where C is the set of frequencies not belonging to the filter transition bands.

- Then, an estimate for F is

$$F \approx b = -\log_2 \left(\sqrt{12} \min_{\omega \in C} \left\{ \left| \frac{\rho(\omega) - ||H(e^{j\omega})| - H_d(e^{j\omega})|}{xS(e^{j\omega})} \right| \right\} \right) \tag{153}$$

Statistical forecast of the wordlength

- This method for estimating the wordlength is also useful in iterative procedures to design filters with minimum wordlength.
- An alternative procedure that is widely used in practice to evaluate the design of digital filters with finite-coefficient wordlength, is to design the filters with tighter specifications than required, quantize the coefficients, and check if the prescribed specifications are still met.
- Obviously, in this case, the success of the design is highly dependent on the designer's experience.

Example 11.8

- Determine the total number of bits required for the filter designed in Example 11.3 to satisfy the following specifications, after coefficient quantization:

$$\left.\begin{aligned} A_p &= 1.2 \text{ dB} \\ A_r &= 39 \text{ dB} \\ \omega_p &= 0.3\pi \text{ rad/sample} \\ \omega_r &= 0.4\pi \text{ rad/sample} \end{aligned}\right\} \tag{154}$$

Example 11.8 - Solution

- Using the specifications in equation (154), we determine

$$\delta_p = 1 - 10^{-A_p/20} = 0.1482 \tag{155}$$

$$\delta_r = 10^{-A_r/20} = 0.0112 \tag{156}$$

and define

$$\rho(\omega) = \begin{cases} \delta_p, & \text{for } 0 \leq \omega \leq 0.3\pi \\ \delta_r, & \text{for } 0.4\pi \leq \omega \leq \pi \end{cases} \tag{157}$$

$$H_d(e^{j\omega}) = \begin{cases} 1, & \text{for } 0 \leq \omega \leq 0.3\pi \\ 0, & \text{for } 0.4\pi \leq \omega \leq \pi \end{cases} \tag{158}$$

Example 11.8 - Solution

- A reasonable certainty margin is about 90%, yielding, from equation (149),

$$\frac{x}{\sqrt{2}} = \texttt{erfinv}(0.9) = 1.1631 \Rightarrow x = 1.6449 \tag{159}$$

- We use the filter designed in Example 11.3 as $H(e^{j\omega})$, with the corresponding sensitivity function $S(e^{j\omega})$, as given in equation (140) and depicted in Figure 24a.

- Based on these values, we can compute the number of bits F for the fractional part, using equation (153), resulting in $F \approx 12.0993$, which we round to $F = 12$ bits.

- From Table 1, we observe that $I = 3$ bits are necessary to represent the integer part of the filter coefficients which fall in the range -4 to $+4$. Therefore, the total number of bits required, including the sign bit, is

$$B = I + F + 1 = 16 \tag{160}$$

Example 11.8 - Solution

- Table 2 shows the filter coefficients after quantization.

Table 2: Quantized filter coefficients for specifications (154).

Numerator coefficients		Denominator coefficients	
$b_0 =$	$0.028\,320\,31$	$a_0 =$	$1.000\,000\,00$
$b_1 =$	$-0.001\,464\,84$	$a_1 =$	$-3.028\,564\,45$
$b_2 =$	$0.031\,738\,28$	$a_2 =$	$4.567\,871\,09$
$b_3 =$	$0.031\,738\,28$	$a_3 =$	$-3.900\,146\,48$
$b_4 =$	$-0.001\,464\,84$	$a_4 =$	$1.896\,728\,52$
$b_5 =$	$0.028\,320\,31$	$a_5 =$	$-0.418\,945\,31$

Example 11.8 - Solution

- Table 3 includes the resulting passband ripple and stopband attenuation for several values of F, from which we can clearly see that using the predicted $\mathrm{F} = 12$, the specifications in equation (154) are satisfied, even after quantization of the filter coefficients.

Table 3: Filter characteristics as a function of the number of fractional bits F.

F	$\mathrm{A_p}$ [dB]	$\mathrm{A_r}$ [dB]
15	1.0100	40.0012
14	1.0188	40.0106
13	1.0174	40.0107
12	**1.1625**	**39.7525**
11	1.1689	39.7581
10	1.2996	39.7650
9	1.2015	40.0280
8	2.3785	40.2212

Limit cycles

- A serious practical problem that affects the implementation of recursive digital filters is the possible occurrence of parasitic oscillations.
- These oscillations can be classified, according to their origin, as either granular or overflow limit cycles, as presented below.

Granular limit cycles

- Any stable digital filter, if implemented with idealized infinite-precision arithmetic, should have an asymptotically decreasing response when the input signal becomes zero after a given instant of time $n_0 T$.
- However, if the filter is implemented with finite-precision arithmetic, the noise signals generated at the quantizers become highly correlated from sample to sample and from source to source.
- This correlation can cause autonomous oscillations, referred to as granular limit cycles, originating from quantization performed in the least significant signal bits, as indicated in the example that follows.

Example 11.9

- Suppose the filter of Figure 26 has the following input signal:

$$x(n) = \begin{cases} 0.111, & \text{for } n = 1 \\ 0.000, & \text{for } n \neq 1 \end{cases} \tag{161}$$

where the numbers are represented in two's complement. Determine the output signal in the case when the quantizer performs rounding, for $n = 1, 2, \ldots, 40$.

Example 11.9

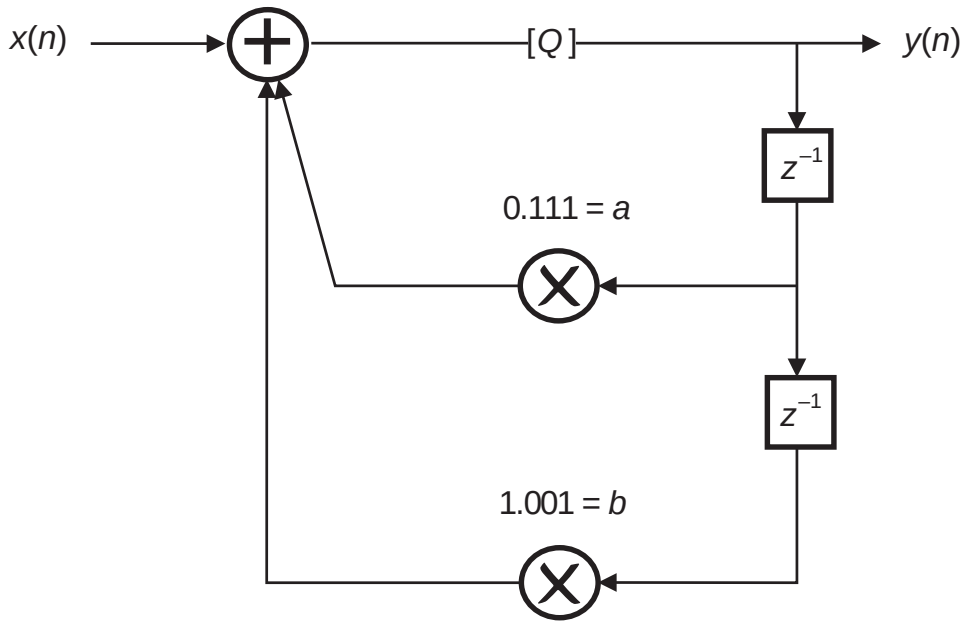


Figure 26: Second-order section with a quantizer.

Example 11.9 - Solution

- The output in the time domain, assuming that the quantizer rounds the signal, is given in Table 4, where one can easily see that an oscillation is sustained at the output, even after the input becomes zero.
- In many practical applications, where the signal levels in a digital filter can be constant or very low, even for short periods of time, limit cycles are highly undesirable, and should be eliminated or at least have their amplitude bounds strictly limited.

Example 11.9 - Solution

Table 4: Output signal of network shown in Figure 26.

n	$y(n)$
1	0.111
2	$Q(0.110\,001 + 0.000\,000) = 0.110$
3	$Q(0.101\,010 + 1.001\,111) = 1.111$
4	$Q(1.111\,001 + 1.010\,110) = 1.010$
5	$Q(1.010\,110 + 0.000\,111) = 1.100$
6	$Q(1.100\,100 + 0.101\,010) = 0.010$
7	$Q(0.001\,110 + 0.011\,100) = 0.101$
8	$Q(0.100\,011 + 1.110\,010) = 0.011$
9	$Q(0.010\,101 + 1.011\,101) = 1.110$
10	$Q(1.110\,010 + 1.101\,011) = 1.100$
11	$Q(1.100\,100 + 0.001\,110) = 1.110$
12	$Q(1.110\,010 + 0.011\,100) = 0.010$
13	$Q(0.001\,110 + 0.001\,110) = 0.100$
14	$Q(0.011\,100 + 1.110\,010) = 0.010$
15	$Q(0.001\,110 + 1.100\,100) = 1.110$
16	$Q(1.110\,010 + 1.110\,010) = 1.100$
17	$Q(1.100\,100 + 0.001\,110) = 1.110$
18	$Q(1.110\,010 + 0.011\,100) = 0.010$
19	$Q(0.001\,110 + 0.001\,110) = 0.100$
20	$Q(0.011\,100 + 1.110\,010) = 0.010$
21	$Q(0.001\,110 + 1.100\,100) = 1.110$
22	$Q(1.110\,010 + 1.110\,010) = 1.100$
⋮	⋮

Overflow limit cycles

- Overflow limit cycles can occur when the magnitudes of the internal signals exceed the available register range.
- In order to avoid the increase of the signal wordlength in recursive digital filters, overflow nonlinearities must be applied to the signal.
- Such nonlinearities influence the most significant bits of the signal, possibly causing severe distortion.
- An overflow can give rise to self-sustained, high-amplitude oscillations, widely known as overflow limit cycles.
- Overflow can occur in any structure in the presence of an input signal, and input-signal scaling is crucial to reduce the probability of overflow to an acceptable level.

Example 11.10

- Consider the filter of Figure 27 with $a = 0.9606$ and $b = 0.9849$, where the overflow nonlinearity employed is the two's complement with 3-bit quantization (see Figure 27).
- Its analytic expression is given by

$$Q(x) = \frac{1}{4}[(\lceil 4x - 0.5 \rceil + 4) \mod 8] - 1 \tag{162}$$

  where $\lceil x \rceil$ means the smallest integer larger than or equal to x.
- Determine the output signal of such a filter for zero input, given the initial conditions $y(-2) = 0.50$ and $y(-1) = -1.00$.

Example 11.10

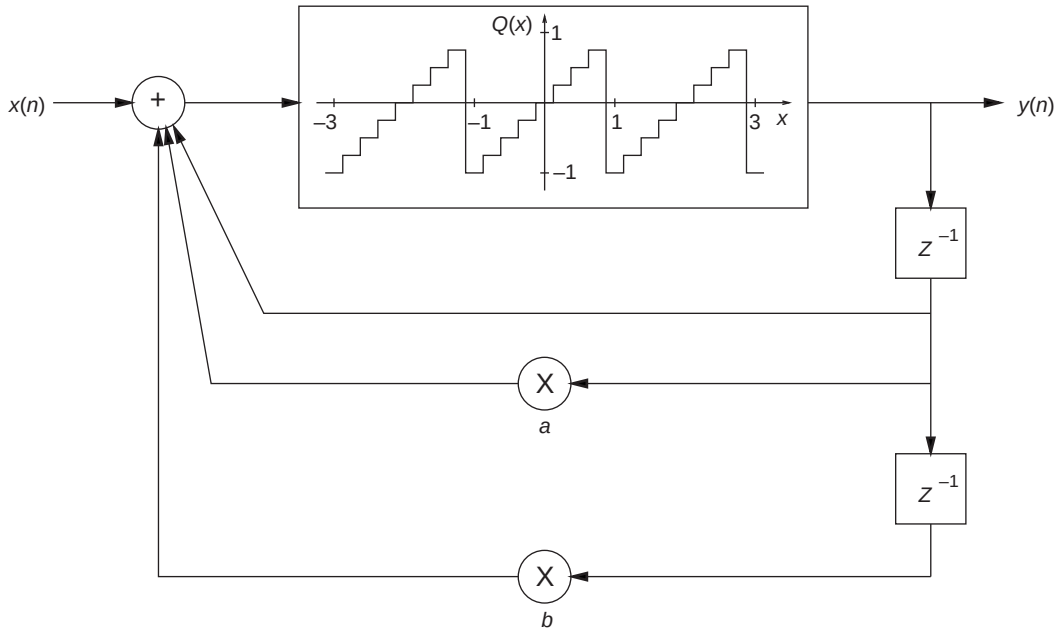


Figure 27: Second-order section with an overflow quantizer.

Example 11.10 - Solution

- With $a = 0.9606$, $b = -0.9849$, $y(-2) = 0.50$ and $y(-1) = -1.00$, we have that

$$\begin{aligned}
y(0) &= Q[1.9606(-1.00) - 0.9849(0.50)] &&= Q[-2.4530] &&= -0.50 \\
y(1) &= Q[1.9606(-0.50) - 0.9849(-1.00)] &&= Q[0.0046] &&= 0.00 \\
y(2) &= Q[1.9606(0.00) - 0.9849(-0.50)] &&= Q[0.4924] &&= 0.50 \\
y(3) &= Q[1.9606(0.50) - 0.9849(0.00)] &&= Q[0.9803] &&= -1.00 \\
&\vdots
\end{aligned} \tag{163}$$

- Since $y(2) = y(-2)$ and $y(3) = y(-1)$, we have that, although there is no excitation, the output signal in nonzero and periodic with a period of 4, thus indicating the existence of overflow limit cycles.

Overflow limit cycles

- A digital filter structure is considered free from overflow limit cycles if the error introduced in the filter after an overflow decreases with time in such a way that the output of the nonlinear filter (including the quantizers) converges to the output of the ideal linear filter.
- In practice a quantizer incorporates nonlinearities corresponding to both granular quantization and overflow.
- Figure 28 illustrates a digital filter using a quantizer that implements rounding as the granular quantization and saturation arithmetic as the overflow nonlinearity.
- Note that although this overflow nonlinearity is different from the one depicted in Figure 27, both are classified as overflow.

Overflow limit cycles

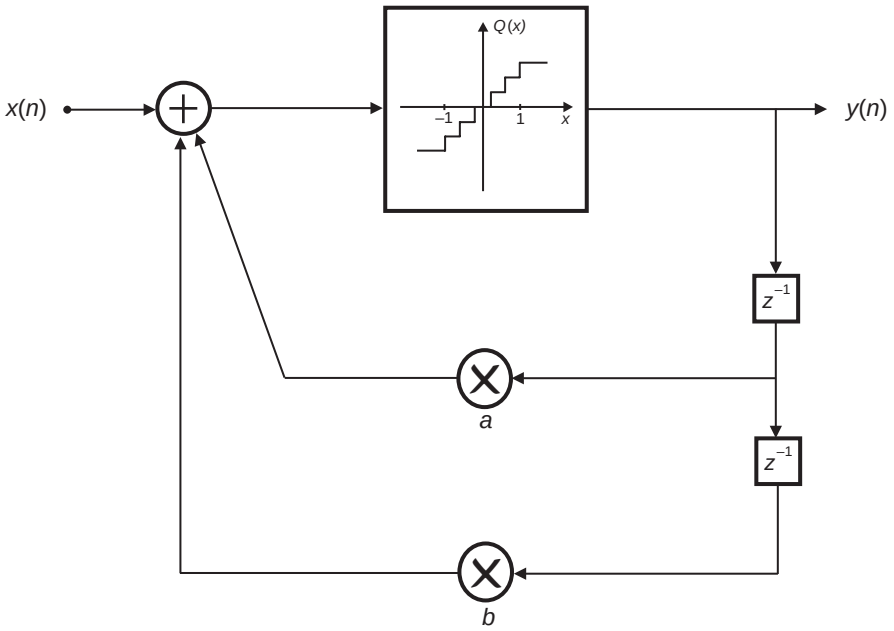


Figure 28: Second-order section with a rounding and saturating quantizer.

Elimination of zero-input limit cycles

- A general IIR filter can be depicted as in Figure 29a, where the linear N-port network consists of interconnections of multipliers and adders.
- In a recursive filter implemented with fixed-point arithmetic, each internal loop contains a quantizer.
- Assuming that the quantizers are placed at the delay inputs (the state variables), as shown in Figure 29b, we can describe the digital filter, including the quantizers, using the following state-space formulation:

$$\begin{aligned} \mathbf{x}(n+1) &= [\mathbf{A}\mathbf{x}(n) + \mathbf{b}u(n)]_Q \\ y(n) &= \mathbf{c}^T\mathbf{x}(n) + du(n) \end{aligned} \tag{164}$$

where $[x]_Q$ indicates the quantized value of x, **A** is the state matrix, **b** is the input vector, **c** is the output vector, and d represents the direct connection between the input and output of the filter.

Elimination of zero-input limit cycles

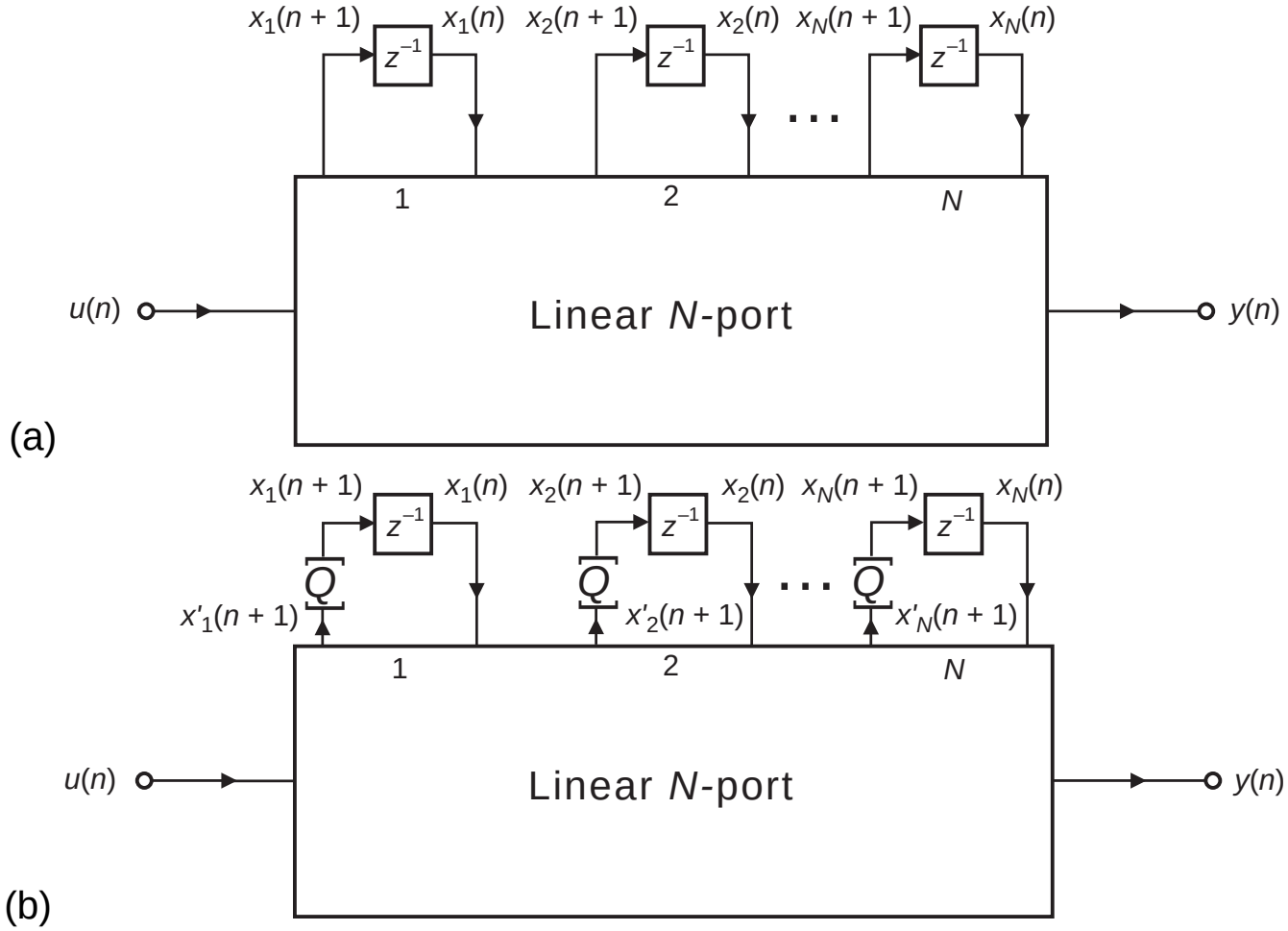


Figure 29: Digital filter networks: (a) ideal; (b) with quantizers at the state variables.

Elimination of zero-input limit cycles

- In order to analyze zero-input limit cycles, it is sufficient to consider the recursive part of the state equation given by

$$\mathbf{x}(k+1) = [\mathbf{A}\mathbf{x}(k)]_Q = [\mathbf{x}'(k+1)]_Q \tag{165}$$

where the quantization operations $[\cdot]_Q$ are nonlinear operations such as truncation, rounding, or overflow.

Elimination of zero-input limit cycles

- The basis for the elimination of nonlinear oscillations is given by Theorem 11.1 below.
- **Theorem:** If a stable digital filter has a state matrix **A** and, for any $\mathrm{N} \times 1$ vector **x** there exists a diagonal positive-definite matrix **G**, such that

$$\hat{\mathbf{x}}^{\mathrm{T}}(\mathbf{G} - \mathbf{A}^{\mathrm{T}}\mathbf{G}\mathbf{A})\hat{\mathbf{x}} \geq 0 \tag{166}$$

  then the granular zero-input limit cycles can be eliminated if the quantization is performed through magnitude truncation.

- **Proof:** Consider a non-negative pseudo-energy Lyapunov function given by

$$\mathrm{p}(\mathbf{x}(\mathrm{n})) = \mathbf{x}^{\mathrm{T}}(\mathrm{n})\mathbf{G}\mathbf{x}(\mathrm{n}) \tag{167}$$

Elimination of zero-input limit cycles

- The energy variation in a single iteration can be defined as

$$
\begin{aligned}
\Delta p(n+1) &= p(\mathbf{x}(n+1)) - p(\mathbf{x}(n)) \\
&= \mathbf{x}^T(n+1)\mathbf{G}\mathbf{x}(n+1) - \mathbf{x}^T(n)\mathbf{G}\mathbf{x}(n) \\
&= [\mathbf{x}'^T(n+1)]_Q \mathbf{G} [\mathbf{x}'(n+1)]_Q - \mathbf{x}^T(n)\mathbf{G}\mathbf{x}(n) \\
&= [\mathbf{A}\mathbf{x}(n)]_Q^T \mathbf{G} [\mathbf{A}\mathbf{x}(n)]_Q - \mathbf{x}^T(n)\mathbf{G}\mathbf{x}(n) \\
&= [\mathbf{A}\mathbf{x}(n)]^T \mathbf{G} [\mathbf{A}\mathbf{x}(n)] - \mathbf{x}^T(n)\mathbf{G}\mathbf{x}(n) \\
&\quad - \sum_{i=1}^{N} (x_i'^2(n+1) - x_i^2(n+1)) g_i \\
&= \mathbf{x}^T(n)[\mathbf{A}^T\mathbf{G}\mathbf{A} - \mathbf{G}]\mathbf{x}(n) - \sum_{i=1}^{N} (x_i'^2(n+1) - x_i^2(n+1)) g_i
\end{aligned}
\tag{9.168}
$$

where g_i are the diagonal elements of $\mathbf{G}$.

Elimination of zero-input limit cycles

- If quantization is performed through magnitude truncation, then the errors due to granular quantization and overflow are such that

$$|x_i(n+1)| \leq |x'_i(n+1)| \tag{169}$$

for all i and n. Therefore, if equation (166) holds, from equation (168), we have that

$$\Delta p(n+1) \leq 0 \tag{170}$$

- If a digital filter is implemented with finite-precision arithmetic, within a finite number of samples after the input signal becomes zero, the output signal will become either a periodic oscillation or zero.

Elimination of zero-input limit cycles

- Periodic oscillations, with nonzero amplitude, can not be sustained if $\Delta p(n+1) \leq 0$, as shown above.
- Therefore, equations (166) and (169) are sufficient conditions to guarantee the elimination of granular zero-input limit cycles on a recursive digital filter.
- Note that the condition given in equation (166) is equivalent to requiring that **F** be positive semidefinite, where

$$\mathbf{F} = (\mathbf{G} - \mathbf{A}^T\mathbf{G}\mathbf{A}) \tag{171}$$

- It is worth observing that for any stable state matrix **A**, its eigenvalues are inside the unit circle, and there will always be a positive-definite and symmetric matrix **G** such that **F** is symmetric and positive semidefinite.

Elimination of zero-input limit cycles

- However, if **G** is not diagonal, the quantization process required to eliminate zero-input limit cycles is extremely complicated, as the quantization operation in each quantizer is coupled to the others.
- On the other hand, if there is a matrix **G** which is diagonal and positive definite such that **F** is positive semidefinite, then zero-input limit cycles can be eliminated by simple magnitude truncation.
- In the following theorem, we will state more specific conditions regarding the elimination of zero-input limit cycles in second-order systems.

Elimination of zero-input limit cycles

- **Theorem:** Given a 2×2 stable state matrix **A**, there is a diagonal positive-definite matrix **G**, such that **F** is positive semidefinite, if and only if

$$a_{12}a_{21} \geq 0 \tag{172}$$

or

$$\left.\begin{aligned} a_{12}a_{21} &< 0 \\ |a_{11} - a_{22}| + \det(\mathbf{A}) &\leq 1 \end{aligned}\right\} \tag{173}$$

Elimination of zero-input limit cycles

- **Proof:** Let $\mathbf{G} = (\mathbf{T}^{-1})^2$ be a diagonal positive-definite matrix, such that $\mathbf{T}$ is a diagonal nonsingular matrix.
- Therefore, we can write $\mathbf{F}$ as

$$\mathbf{F} = \mathbf{T}^{-1}\mathbf{T}^{-1} - \mathbf{A}^{\mathsf{T}}\mathbf{T}^{-1}\mathbf{T}^{-1}\mathbf{A} \tag{174}$$

and then

$$\begin{aligned}\mathbf{T}^{\mathsf{T}}\mathbf{F}\mathbf{T} &= \mathbf{T}^{\mathsf{T}}\mathbf{T}^{-1}\mathbf{T}^{-1}\mathbf{T} - \mathbf{T}^{\mathsf{T}}\mathbf{A}^{\mathsf{T}}\mathbf{T}^{-1}\mathbf{T}^{-1}\mathbf{A}\mathbf{T} \\ &= \mathbf{I} - (\mathbf{T}^{-1}\mathbf{A}\mathbf{T})^{\mathsf{T}}(\mathbf{T}^{-1}\mathbf{A}\mathbf{T}) \\ &= \mathbf{I} - \mathbf{M}\end{aligned} \tag{175}$$

with $\mathbf{M} = (\mathbf{T}^{-1}\mathbf{A}\mathbf{T})^{\mathsf{T}}(\mathbf{T}^{-1}\mathbf{A}\mathbf{T})$, as $\mathbf{T}^{\mathsf{T}} = \mathbf{T}$.

- Since the matrix $(\mathbf{I} - \mathbf{M})$ is symmetric and real, its eigenvalues are real.

Elimination of zero-input limit cycles

- This matrix is then positive semidefinite if and only if its eigenvalues are non-negative, or, equivalently, if and only if its trace and determinant are non-negative. We then have that

$$\det\{\mathbf{I}-\mathbf{M}\not{+}1\} + \det\{\mathbf{M}\} - \mathrm{tr}\{\mathbf{M}\} = 1 + (\det\{\mathbf{A}\})^2 - \mathrm{tr}\{\mathbf{M}\} \tag{176}$$

$$\mathrm{tr}\{\mathbf{I}-\mathbf{M}\not{+}2 - \mathrm{tr}\{\mathbf{M}\} \tag{177}$$

- For a stable digital filter, it is easy to verify that $\det\{\mathbf{A}\} < 1$, and then

$$\mathrm{tr}\{\mathbf{I}-\mathbf{M}\} > \det\{\mathbf{I}-\mathbf{M}\} \tag{178}$$

- Hence, the condition $\det\{\mathbf{I}-\mathbf{M}\} \geq 0$ is necessary and sufficient to guarantee that $(\mathbf{I}-\mathbf{M})$ is positive semidefinite.

Elimination of zero-input limit cycles

- From the definition of **M**, and using $\alpha = t_{22}/t_{11}$, then

$$\det\{\mathbf{I} - \mathbf{M}\} = 1 + (\det\{\mathbf{A}\})^2 - \left(a_{11}^2 + \alpha^2 a_{12}^2 + \frac{a_{21}^2}{\alpha^2} + a_{22}^2 \right) \tag{179}$$

- By calculating the maximum of the equation above with respect to α, we get an optimal $\alpha^\star$ such that

$$(\alpha^\star)^2 = \left| \frac{a_{21}}{a_{12}} \right| \tag{180}$$

and then

$$\begin{aligned} \det{}^\star\{\mathbf{I} - \mathbf{M}\} &= 1 + (\det\{\mathbf{A}\})^2 - (a_{11}^2 + 2|a_{12}a_{21}| + a_{22}^2) \\ &= (1 + \det\{\mathbf{A}\})^2 - (\mathrm{tr}\{\mathbf{A}\})^2 + 2(a_{12}a_{21} - |a_{12}a_{21}|) \end{aligned} \tag{181}$$

where $\det^\star$ denotes the maximum value of the respective determinant.

Elimination of zero-input limit cycles

- We now analyze two separate cases to guarantee that $\det^{\star}\{\mathbf{I}-\mathbf{M}\} \geq 0$.
  - If

$$a_{12}a_{21} \geq 0 \tag{182}$$

then

$$\begin{aligned}\det^{\star}\{\mathbf{I}-\mathbf{M}\} &= (1+\det\{\mathbf{A}\})^2 - (\mathrm{tr}\{\mathbf{A}\})^2 \\ &= (1+\alpha_2)^2 - (-\alpha_1)^2 \\ &= (1+\alpha_1+\alpha_2)(1-\alpha_1+\alpha_2)\end{aligned} \tag{183}$$

where $\alpha_1 = -\mathrm{tr}\{\mathbf{A}\}$ and $\alpha_2 = \det\{\mathbf{A}\}$ are the filter denominator coefficients. It can be verified that, for a stable filter, $(1+\alpha_1+\alpha_2)(1-\alpha_1+\alpha_2) > 0$, and then equation (182) implies that $(\mathbf{I}-\mathbf{M})$ is positive definite.

Elimination of zero-input limit cycles

- (cont.)
  - If

$$a_{12}a_{21} < 0 \tag{184}$$

then

$$\begin{aligned} \det{}^{\star}\{\mathbf{I}-\mathbf{M}\} &= 1 + (\det\{\mathbf{A}\})^2 - (a_{11}^2 - 2a_{12}a_{21} + a_{22}^2) \\ &= (1 - \det\{\mathbf{A}\})^2 - (a_{11} - a_{22})^2 \end{aligned} \tag{185}$$

This equation is greater than or equal to zero, if and only if

$$|a_{11} - a_{22}| + \det\{\mathbf{A}\} \leq 1 \tag{186}$$

Elimination of zero-input limit cycles

- Therefore, either equation (182) or equations (184) and (186) are the necessary and sufficient conditions for the existence of a diagonal matrix

$$\mathbf{T} = \text{diag}\{t_{11}\ t_{22}\} \tag{187}$$

with

$$\frac{t_{22}}{t_{11}} = \sqrt{\left|\frac{a_{21}}{a_{12}}\right|} \tag{188}$$

such that **F** is positive semidefinite.

Elimination of zero-input limit cycles

- It is worth observing that the above theorem gives the conditions for the matrix **F** to be positive semidefinite for second-order sections.
- In the example below, we illustrate the limit-cycle elimination process by showing, without resorting to Theorem 11.2, that a given second-order structure is free from zero-input limit cycles.
- The reader is encouraged to apply the theorem to show the same result.

Example 11.11

- Examine the possibility of eliminating limit cycles in the network of Figure 30.

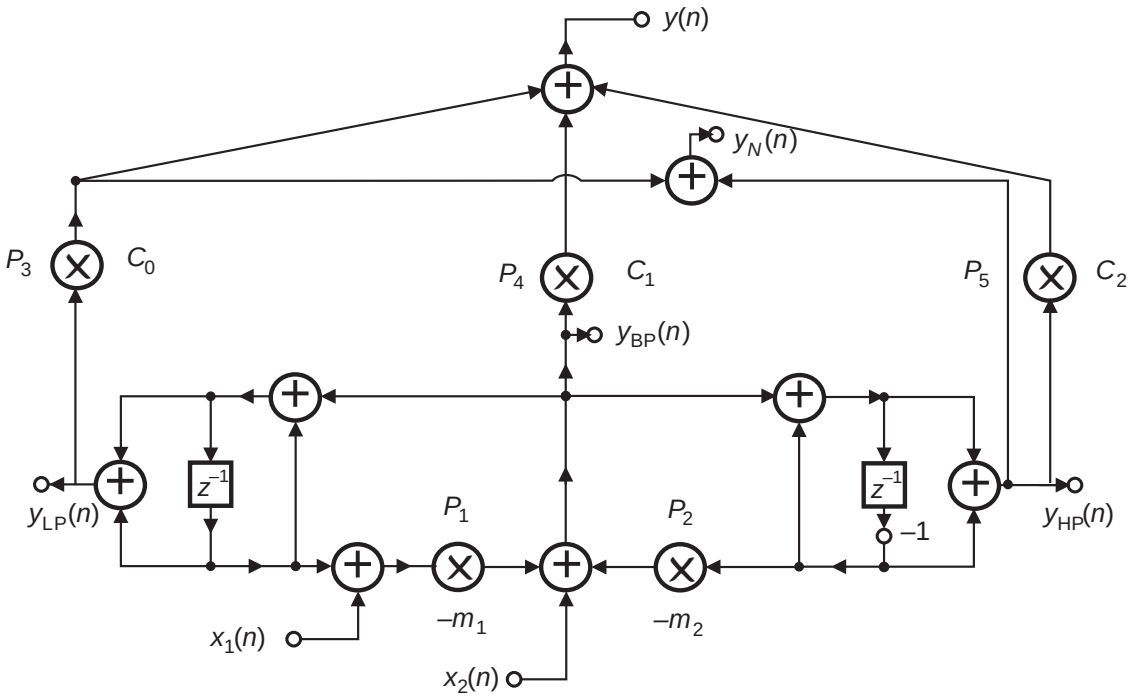


Figure 30: General-purpose network.

Example 11.11 - Solution

- The structure in Figure 30 realizes lowpass, bandpass, and highpass transfer functions simultaneously (with subscripts LP, BP, and HP, respectively).
- The structure also realizes a transfer function with zeros on the unit circle, using the minimum number of multipliers.
- The characteristic polynomial of the structure is given by

$$D(z) = z^2 + (m_1 - m_2)z + m_1 + m_2 - 1 \tag{189}$$

- In order to guarantee stability, the multiplier coefficients m_1 and m_2 should fall in the range

$$\left.\begin{aligned} m_1 &> 0 \\ m_2 &> 0 \\ m_1 + m_2 &< 2 \end{aligned}\right\} \tag{190}$$

Example 11.11 - Solution

- Figure 31 depicts the recursive part of the structure in Figure 30, including the quantizers.

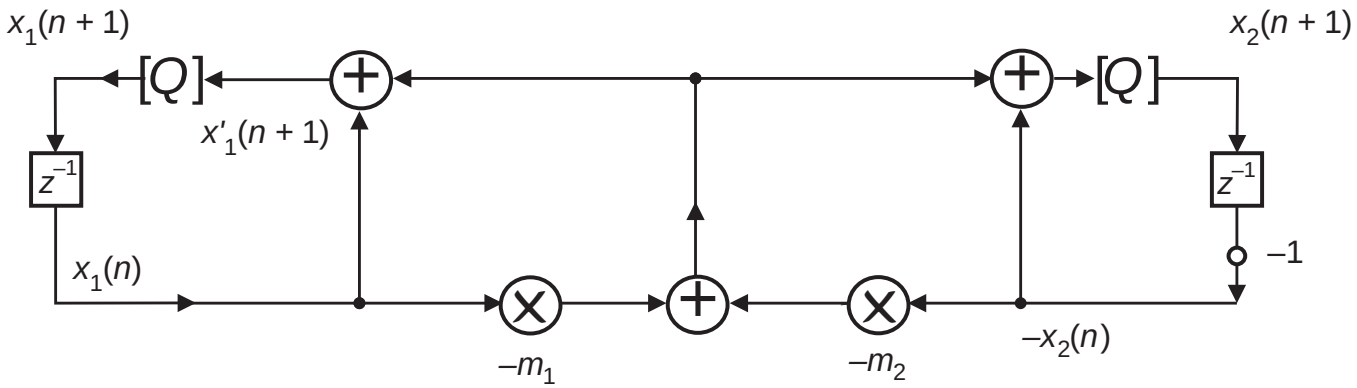


Figure 31: Recursive part of the network in Figure 30.

Example 11.11 - Solution

- The zero-input state-space equation for the structure in Figure 31 is

$$\mathbf{x}'(n+1) = \begin{bmatrix} x_1'(n+1) \\ x_2'(n+1) \end{bmatrix} = \mathbf{A} \begin{bmatrix} x_1(n) \\ x_2(n) \end{bmatrix} \tag{191}$$

with

$$\mathbf{A} = \begin{bmatrix} (1-m_1) & m_2 \\ -m_1 & (m_2-1) \end{bmatrix} \tag{192}$$

- By applying quantization to $\mathbf{x}'(n+1)$, we find

$$\mathbf{x}(n+1) = [\mathbf{x}'(n+1)]_Q = [\mathbf{A}\mathbf{x}(n)]_Q \tag{193}$$

Example 11.11 - Solution

- A quadratic positive-definite function can be defined as

$$p(\mathbf{x}(n)) = \mathbf{x}^T(n)\mathbf{G}\mathbf{x}(n) = \frac{x_1^2}{m_2} + \frac{x_2^2}{m_1} \tag{194}$$

with

$$\mathbf{G} = \begin{bmatrix} \frac{1}{m_2} & 0 \\ 0 & \frac{1}{m_1} \end{bmatrix} \tag{195}$$

which is positive definite, since from equation (190), $m_1 > 0$ and $m_2 > 0$.

Example 11.11 - Solution

- An auxiliary energy increment is then

$$\begin{aligned}
\Delta p_0(n+1) &= p(\mathbf{x}'(n+1)) - p(\mathbf{x}(n)) \\
&= \mathbf{x}'^{T}(n+1)\mathbf{G}\mathbf{x}'(n+1) - \mathbf{x}^{T}(n)\mathbf{G}\mathbf{x}(n) \\
&= \mathbf{x}^{T}(n)[\mathbf{A}^{T}\mathbf{G}\mathbf{A} - \mathbf{G}]\mathbf{x}(n) \\
&= (m_1 + m_2 - 2)\left(x_1(n)\sqrt{\frac{m_1}{m_2}} - x_2(n)\sqrt{\frac{m_2}{m_1}}\right)^2 \qquad (196)
\end{aligned}$$

Example 11.11 - Solution

- Since from equation (190), $m_1 + m_2 < 2$, then

$$\left.\begin{aligned} \Delta p_0(n+1) = 0, \quad \text{for } x_1(n) = x_2(n)\frac{m_2}{m_1} \\ \Delta p_0(n+1) < 0, \quad \text{for } x_1(n) \neq x_2(n)\frac{m_2}{m_1} \end{aligned}\right\} \tag{197}$$

- Now, if magnitude truncation is applied to quantize the state variables, then $p(\mathbf{x}(n)) \leq p(\mathbf{x}'(n))$, which implies that

$$\Delta p(\mathbf{x}(n)) = p(\mathbf{x}(n+1)) - p(\mathbf{x}(n)) \leq 0 \tag{198}$$

  and then $p(\mathbf{x}(n))$ is a Lyapunov function.

- Overall, when no quantization is applied in the structure of Figure 30, no self-sustained oscillations occur if the stability conditions of equation (190) are satisfied.

Example 11.11 - Solution

- If, however, quantization is applied to the structure, as shown in Figure 31, oscillations may occur.
- Using magnitude truncation, then $|x_i(n)| \leq |x'_i(n)|$, and under these circumstances, $p(\mathbf{x}(n))$ decreases during the subsequent iterations, and eventually the oscillations disappear, with

$$\mathbf{x}(n) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{199}$$

being the only possible equilibrium point.

Elimination of constant-input limit cycles

- As seen above, the sufficient conditions for the elimination of zero-input limit cycles are well established.
- However, if the system input is a nonzero constant, limit cycles may still occur.
- It is worth observing that the response of a stable linear system to a constant input signal should also be a constant signal.
- In the associated literature, a theorem is presented that establishes how constant-input limit cycles can also be eliminated in digital filters in which zero-input limit cycles are eliminated. This theorem is as follows.

Elimination of constant-input limit cycles

- **Theorem:** Assume that the general digital filter in Figure 29b does not sustain zero-input limit cycles and that

$$\left.\begin{aligned} \mathbf{x}(n+1) &= [\mathbf{A}\mathbf{x}(n) + \mathbf{B}u(n)]_Q \\ y(n) &= \mathbf{C}^T\mathbf{x}(n) + du(n) \end{aligned}\right\} \tag{200}$$

- Constant-input limit cycles can also be eliminated, by modifying the structure in Figure 29b, as shown in Figure 32, where

$$\mathbf{p} = [p_1\ p_2 \cdots p_n]^T = (\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} \tag{201}$$

and $\mathbf{p}u_0$ must be representable in the machine wordlength, where u_0 is a constant input signal.

Elimination of constant-input limit cycles

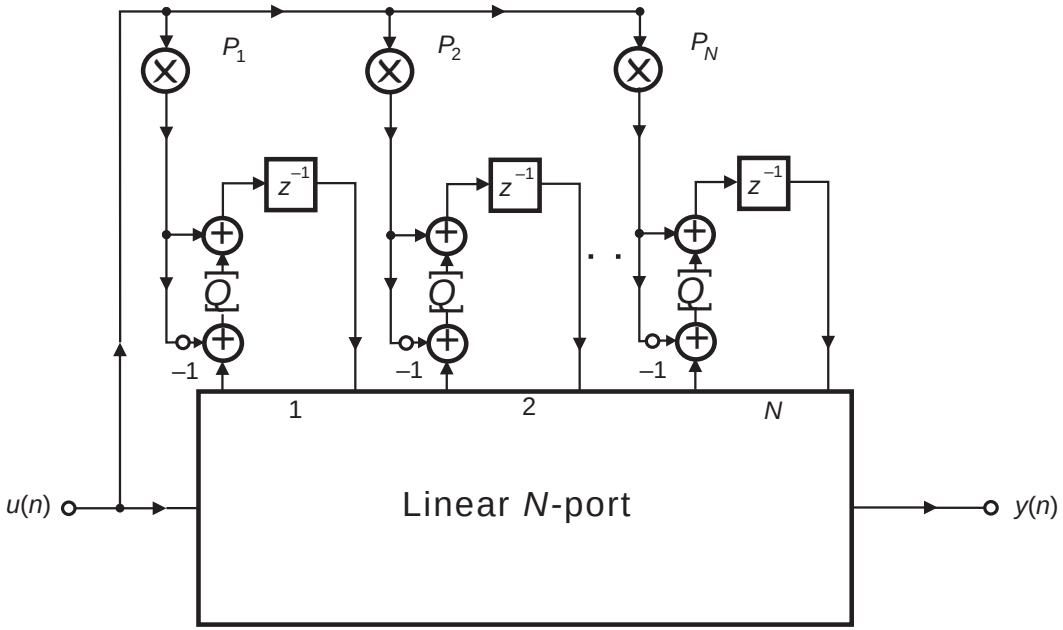


Figure 32: Modified Nth-order network for the elimination of constant-input limit cycles.

Elimination of constant-input limit cycles

- **Proof:** Since the structure of Figure 29b is free from zero-input limit cycles, the autonomous system

$$\mathbf{x}(n+1) = [\mathbf{A}\mathbf{x}(n)]_Q \tag{202}$$

is such that

$$\lim_{n\to\infty} \mathbf{x}(n) = [0\ 0\ \ldots\ 0]^T \tag{203}$$

- If $\mathbf{p}$ is as given in equation (201), the modified structure of Figure 32 is described by

$$\begin{aligned}\mathbf{x}(n+1) &= [\mathbf{A}\mathbf{x}(n) - \mathbf{p}u_0 + \mathbf{B}u_0]_Q + \mathbf{p}u_0 \\ &= [\mathbf{A}\mathbf{x}(n) - \mathbf{I}(\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}u_0 + (\mathbf{I}-\mathbf{A})(\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}u_0]_Q + \mathbf{p}u_0 \\ &= [\mathbf{A}(\mathbf{x}(n) - \mathbf{p}u_0)]_Q + \mathbf{p}u_0 \end{aligned} \tag{204}$$

Elimination of constant-input limit cycles

- Defining

$$\hat{\mathbf{x}}(n) = \mathbf{x}(n) - \mathbf{p}u_0 \tag{205}$$

then, from equation (204), we can write that

$$\hat{\mathbf{x}}(n+1) = [\mathbf{A}\hat{\mathbf{x}}(n)]_Q \tag{206}$$

- Which is the same as equation (202), except for the transformation in the state variable. Hence, as $\mathbf{p}u_0$ is machine representable (that is, $\mathbf{p}u_0$ can be exactly calculated with the available wordlength), equation (206) also represents a stable system free from constant-input limit cycles.

- If the quantization of the structure depicted in Figure 29b is performed using magnitude truncation, the application of the strategy of Theorem 11.3 leads to the so-called controlled rounding method.

Elimination of constant-input limit cycles

- The constraints imposed by requiring that $\mathbf{p}\mathfrak{u}_0$ be machine representable reduce the number of structures in which the technique described by Theorem 11.3 applies.
- However, there are a large number of second-order sections and wave digital filter structures in which these requirements are automatically met.
- In fact, a large number of research papers have been published proposing new structures which are free from zero-input limit cycles, and free from constant-input limit cycles.
- However, the analysis procedures for the generation of these structures are not unified and here we have aimed to provide a unified framework leading to a general procedure to generate structures which are free from granular limit cycles.

Example 11.12

- Show that by placing the input signal at the point denoted by $x_1(n)$, the structure in Figure 30 is free from constant-input limit cycles.

Example 11.12 - Solution

- The second-order section of Figure 30, with a constant input $x_1(n) = u_0$, can be described by

$$\mathbf{x}(n+1) = \mathbf{A}\mathbf{x}(n) + \begin{bmatrix} -m_1 \\ -m_1 \end{bmatrix} u_0 \tag{207}$$

with **p** such that

$$\mathbf{p} = \begin{bmatrix} m_1 & -m_2 \\ m_1 & 2-m_2 \end{bmatrix}^{-1} \begin{bmatrix} -m_1 \\ -m_1 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \tag{208}$$

- Therefore, $\mathbf{p}u_0$ is clearly machine representable, for any u_0, and the constant-input limit cycles can be eliminated, as depicted in Figure 33.

Example 11.12 - Solution

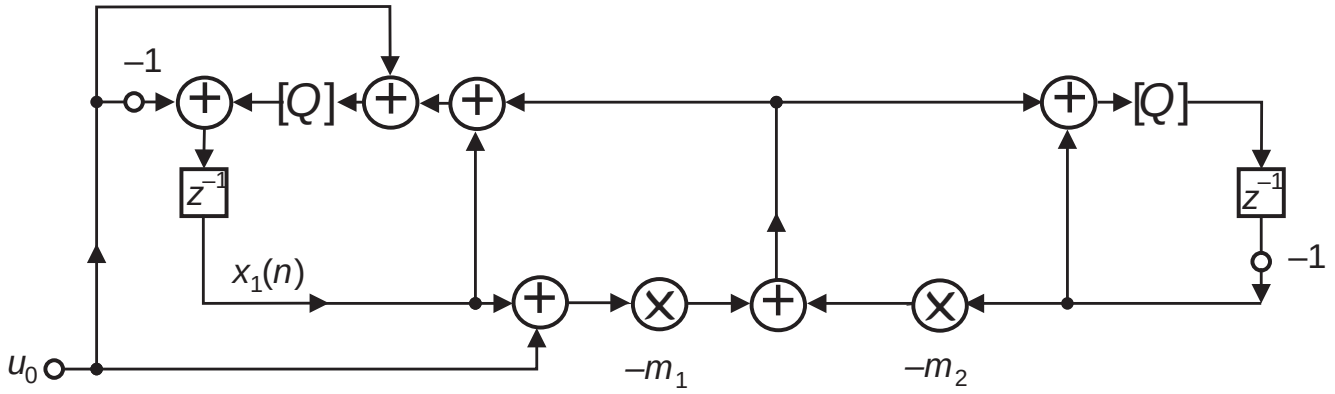


Figure 33: Elimination of constant-input limit cycles in the structure of Figure 30.

Example 11.13

- For the second-order state-variable realization as given in Figure 4.23 of the book, discuss the elimination of constant-input limit-cycles in a distributed arithmetic implementation as the one in Section 11.4.

Example 11.13 - Solution

- In a regular implementation, as the one in Figure 11, to eliminate zero-input limit cycles in the state-space realization, the state variables $x_1(n)$ and $x_2(n)$ must be calculated by ALU_1 and ALU_2 in double precision, and then properly quantized, before being loaded into the shift-registers SR_2 and SR_3.
- However, it can be shown that no double-precision computation is necessary to avoid zero-input limit cycles when implementing the state-space realization with the distributed arithmetic approach.
- To eliminate constant-input limit cycles, the state-space realization shown in Figure 34 requires that the state variable $x_1(n)$ must be computed by ALU_1 in double precision, and then properly quantized to be subtracted from the input signal.

Example 11.13 - Solution

- To perform this subtraction, register A in Figure 9 must be multiplexed with another register that contains the input signal $x(n)$, in order to guarantee that the signal arriving at the adder, at the appropriate instant of time, is the complemented version of $x(n)$, instead of a signal coming from the memory.
- In such a case, the content of the ROM of ALU_1 must be generated as

$$s'_{1j} = a_{11}x_{1j}(n) + a_{12}x_{2j}(n) + a_{11}x_j(n); \text{ for the ROM of the } ALU_1 \quad (209)$$

while ALU_3 is filled in the same fashion as given in equation (52) and for ALU_2 the content is the same as in equation (51) with b_2 replaced by a_{21}.

Example 11.13 - Solution

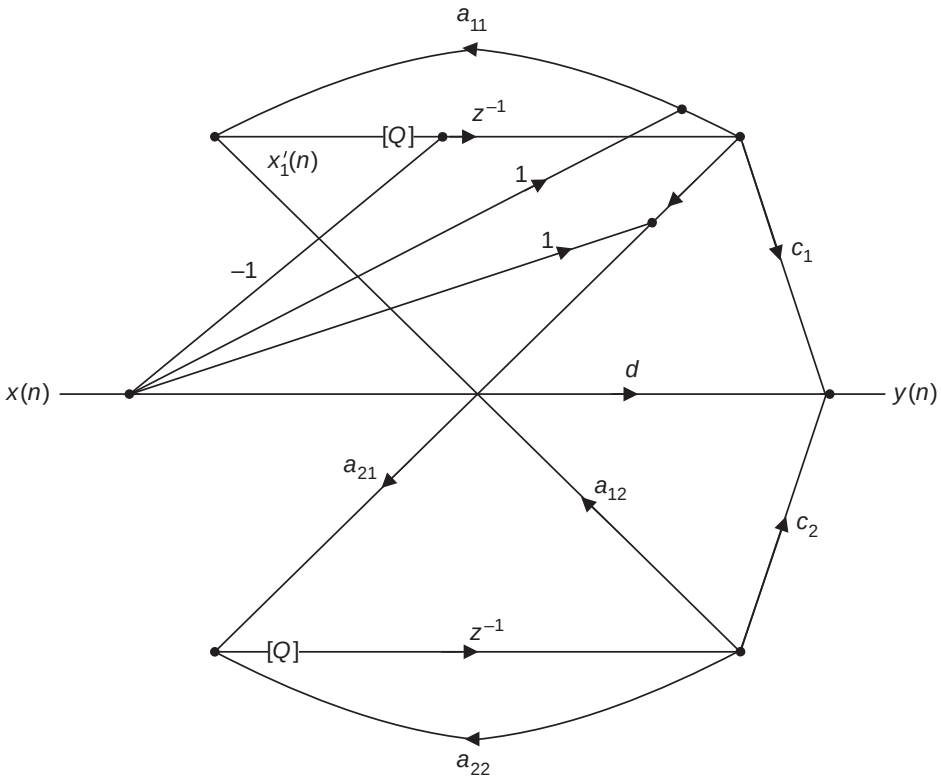


Figure 34: State-space realization immune to constant-input limit cycles.

Forced-response stability of digital filters with nonlinearities due to overflow

- The stability analysis of the forced response of digital filters that include nonlinearities to control overflow must be performed considering input signals for which in the ideal linear system the overflow level is never reached after a given instant n_0.
- In this way, we can verify whether the real system output will recover after an overflow has occurred before instant n_0.
- Although the input signals considered are in a particular class of signals, it can be shown that if the real system recovers for these signals, it will also recover after each overflow, for any input signal, if the recovery period is shorter than the time between two consecutive overflows.
- Consider the ideal linear system as depicted in Figure 35a and the real nonlinear system as depicted in Figure 35b.

Forced-response stability of digital filters with nonlinearities due to overflow

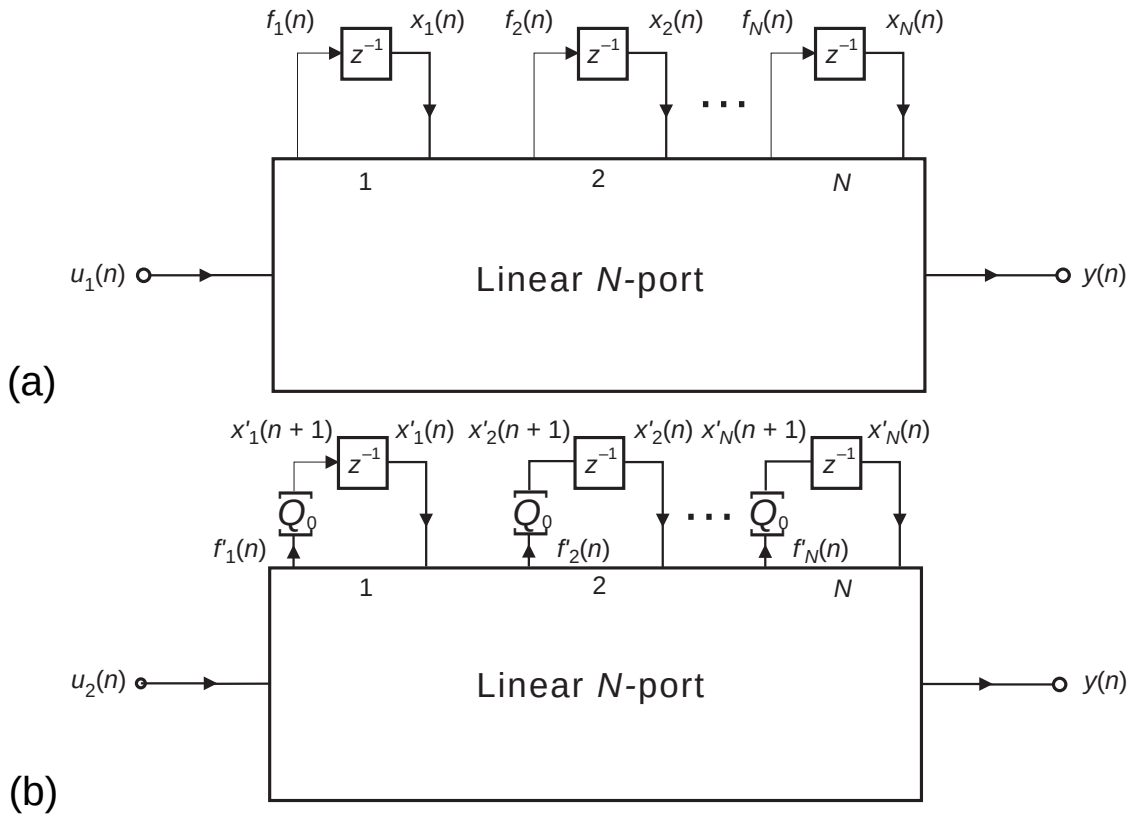


Figure 35: General digital filter networks: (a) ideal; (b) with quantizers at the state variables.

Forced-response stability of digital filters with nonlinearities due to overflow

- The linear system illustrated in Figure 35a is described by the equations

$$\mathbf{f}(n) = \mathbf{A}\mathbf{x}(n) + \mathbf{B}u_1(n) \tag{210}$$

$$\mathbf{x}(n) = \mathbf{f}(n-1) \tag{211}$$

  and the nonlinear system illustrated in Figure 35b is described by the equations

$$\mathbf{f}'(n) = \mathbf{A}\mathbf{x}'(n) + \mathbf{B}u_2(n) \tag{212}$$

$$\mathbf{x}'(n) = [\mathbf{f}'(n-1)]_{Q_o} \tag{213}$$

  where $[u]_{Q_o}$ denotes quantization of u, in the case where an overflow occurs.

- We assume that the output signal of the nonlinear system is properly scaled so that no oscillation due to overflow occurs if it does not occur at the state variables.

Forced-response stability of digital filters with nonlinearities due to overflow

- The response of the nonlinear system of Figure 35b is stable if, when $\mathbf{u}_1(n) = \mathbf{u}_2(n)$, the difference between the outputs of the linear N-port system of Figure 35a, $\mathbf{f}(n)$, and the outputs of the linear N-port system of Figure 35b, $\mathbf{f}'(n)$, tends to zero as $n \to \infty$.

- In other words, if we define an error signal $\mathbf{e}(n) = \mathbf{f}'(n) - \mathbf{f}(n)$, then

$$\lim_{n \to \infty} \mathbf{e}(n) = [0\ 0 \ldots 0]^T \tag{214}$$

- If the difference between the output signals of the two linear N-port systems converges to zero, this implies that the difference between the state variables of both systems will also tend to zero.

Forced-response stability of digital filters with nonlinearities due to overflow

- This can be deduced from equations (210) and (212), which yield

$$\mathbf{e}(n) = \mathbf{f}'(n) - \mathbf{f}(n) = \mathbf{A}[\mathbf{x}'(n) - \mathbf{x}(n)] = \mathbf{A}\mathbf{e}'(n) \tag{215}$$

  where $\mathbf{e}'(n) = \mathbf{x}'(n) - \mathbf{x}(n)$ is the difference between the state variables of both systems.

- Equation (215) is equivalent to saying that $\mathbf{e}(n)$ and $\mathbf{e}'(n)$ are the output and input signals of a linear N-port system described by matrix $\mathbf{A}$, which is the transition matrix of the original system.

- Then, from equation (214), the forced-response stability of the system in Figure 35b is equivalent to the zero-input response of the same system, regardless of the quantization characteristics $[\cdot]_{Q_0}$.

Forced-response stability of digital filters with nonlinearities due to overflow

- Substituting equations (211) and (213) in equation (215), we have that

$$\mathbf{e}'(n) = [\mathbf{f}'(n-1)]_{Q_0} - \mathbf{f}(n-1) = [\mathbf{e}(n-1) + \mathbf{f}(n-1)]_{Q_0} - \mathbf{f}(n-1) \quad (216)$$

- By defining the time-varying vector $\mathbf{v}(\mathbf{e}(n), n)$ as

$$\mathbf{v}(\mathbf{e}(n), n) = [\mathbf{e}(n) + \mathbf{f}(n)]_{Q_0} - \mathbf{f}(n) \quad (217)$$

  equation (216) can be rewritten as

$$\mathbf{e}'(n) = \mathbf{v}(\mathbf{e}(n-1), (n-1)) \quad (218)$$

- The nonlinear system described by equations (215)–(218) is depicted in Figure 36.

Forced-response stability of digital filters with nonlinearities due to overflow

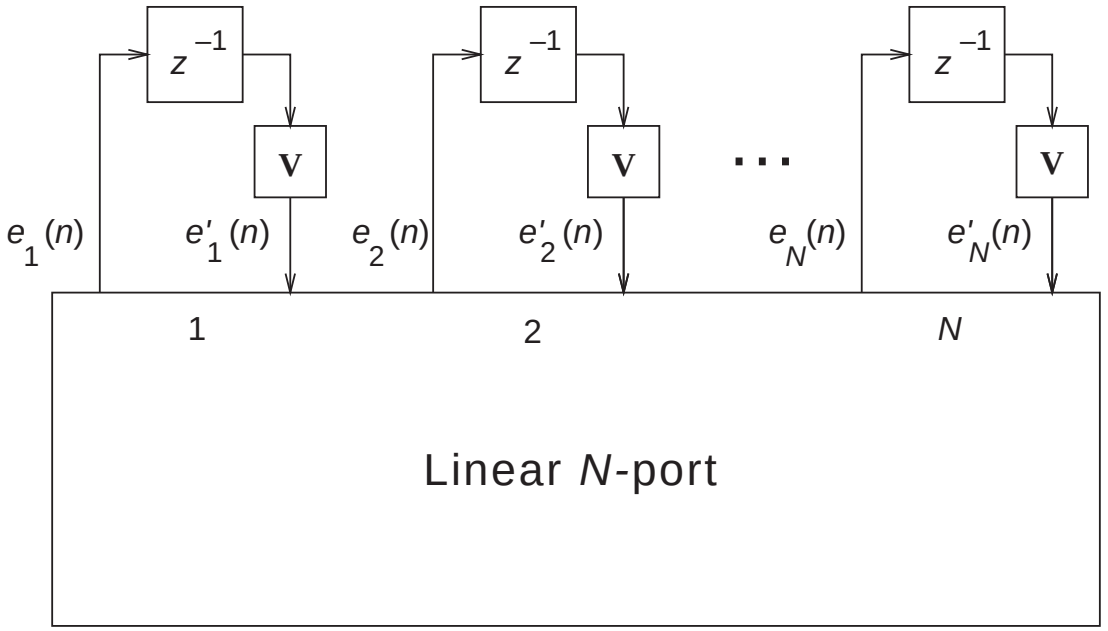


Figure 36: Nonlinear system relating the signals $\mathbf{e}'(n)$ and $\mathbf{e}(n)$.

Forced-response stability of digital filters with nonlinearities due to overflow

- As we saw in Subsection 11.8.3, a system such as the one in Figure 36 is free from zero-input nonlinear oscillations if the nonlinearity $\mathbf{v}(\cdot, n)$ is equivalent to the magnitude truncation, that is

$$|\mathbf{v}(e_i(n), n)| < |e_i(n)|, \quad \text{for } i = 1, 2, \ldots, N \tag{219}$$

- If we assume that the internal signals are such that $|f_i(n)| \leq 1$, for $n > n_0$, then it can be shown that equation (219) remains valid whenever the quantizer Q_0 has overflow characteristics within the hatched region of Figure 37.

Forced-response stability of digital filters with nonlinearities due to overflow

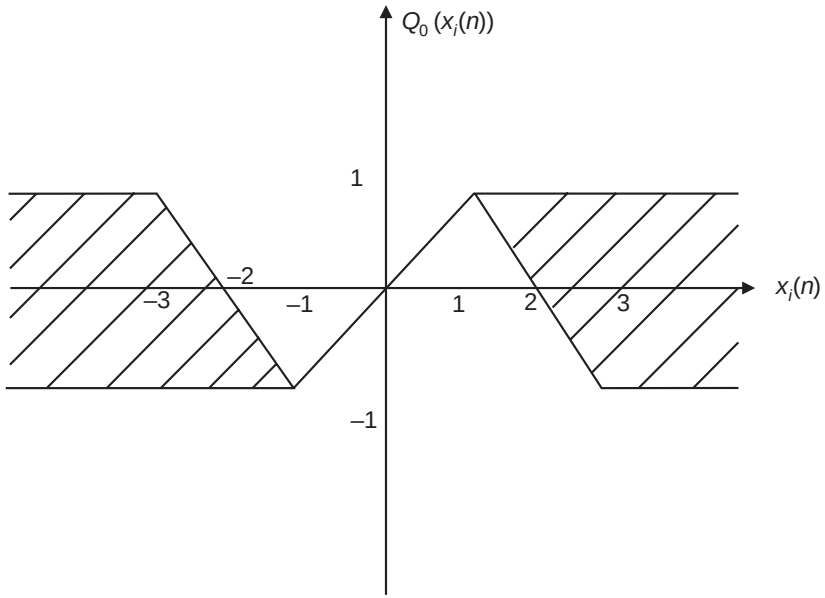


Figure 37: Region for the overflow nonlinearity which guarantees forced-response stability in networks which satisfy Theorem 11.1.

Forced-response stability of digital filters with nonlinearities due to overflow

- Figure 37 can be interpreted as follows:
  - If $-1 \leq x_i(n) \leq 1$, then there should be no overflow.
  - If $1 \leq x_i(n) \leq 3$, then the overflow nonlinearity should be such that $2 - x_i(n) \leq Q_0(x_i(n)) \leq 1$.
  - If $-3 \leq x_i(n) \leq -1$, then the overflow nonlinearity should be such that $-1 \leq Q_0(x_i(n)) \leq -2 - x_i(n)$.
  - If $x_i(n) \geq 3$ or $x_i(n) \leq -3$, then $-1 \leq Q_0(x_i(n)) \leq 1$.
- It is important to note that the overflow nonlinearity of the saturation type (equation (162)) satisfies the requirements in Figure 37.
- Summarizing the above reasoning, one can state that a digital filter which is free of zero-input limit cycles, according to the condition of equation (166), is also forced-input stable, provided that the overflow nonlinearities are in the hatched regions of Figure 37.

Do-it-yourself: Finite-precision digital signal processing

- **Experiment 11.1:** Let us play around digital representation in MATLAB. We concentrate our efforts here in the case $-1 < x < 0$, which yields different $(n+1)$-bit standard, one's-complement, two's-complement, and CSD representations.

- The standard sign-magnitude binary representation `xbin` can be obtained making $s_x = 1$ and following the procedure performed in equation (12), such that

```
x = abs(x); xbin = [1 zeros(1,n)];
for i=2:n+1,
   x = 2*x;
   if x >= 1,
      xbin(i) = 1; x = x-1;
   end;
end;
```

Do-it-yourself: Finite-precision digital signal processing

- The one's-complement `xbin1` representation of x can be determined as

  ```
  xbin1 = [1 ~xbin(2:n+1)];
  ```

  where the $\tilde{x}$ operator determines the binary complement of x in MATLAB.

- For the two's-complement representation `xbin2`, we must add 1 to the least significant bit of `xbin1`. This can be performed, for instance, by detecting the last 0-bit in `xbin1`, which indicates the final position of the carry-over bit, such that

  ```
  xbin2 = xbin1;
  b = max(find(xbin1 == 0));
  xbin2(b:n+1) = ~xbin2(b:n+1);
  ```

Do-it-yourself: Finite-precision digital signal processing

- We can then obtain the CSD representation `xCSD` from `xbin2`, following the algorithm described in Subsection 11.2.2:

```
delta = zeros(1,n+2); theta = zeros(1,n+1); xCSD =
theta;
x2aux = [xbin2(1) xbin2 0];
for i = n:-1:1,
   theta(i) = xor(x2aux(i+1),x2aux(i+2));
   delta(i) = and(~delta(i+1),theta(i));
   xCSD(i) = (1-2*x2aux(i))*delta(i);
end;
```

Do-it-yourself: Finite-precision digital signal processing

- Using $\mathfrak{n} = 7$ and $\mathfrak{x} = -0.6875$ with the scripts above, results in the numerical representations seen in Table 5.

Table 5: Numerical 8-digit representations of $\mathfrak{x} = -0.6875$ in Experiment 11.1.

Numerical Format	$[\mathfrak{x}]$
Standard binary	1.1011000
One's complement	1.0100111
Two's complement	1.0101000
CSD	$\bar{1}$0101000

Do-it-yourself: Finite-precision digital signal processing

- **Experiment 11.2:** Consider the digital filter structure depicted in Figure 38, whose state-space description is given by

$$\mathbf{x}(n+1) = \begin{bmatrix} (1-m_1) & -m_2 \\ m_1 & (m_2-1) \end{bmatrix} \mathbf{x}(n) + \begin{bmatrix} (2-m_1-m_2) \\ -(2-m_1-m_2) \end{bmatrix} u(n) \quad (220)$$

$$y(n) = \begin{bmatrix} -m_1 & -m_2 \end{bmatrix} \mathbf{x}(n) + (1-m_1-m_2)u(n) \quad (221)$$

Do-it-yourself: Finite-precision digital signal processing

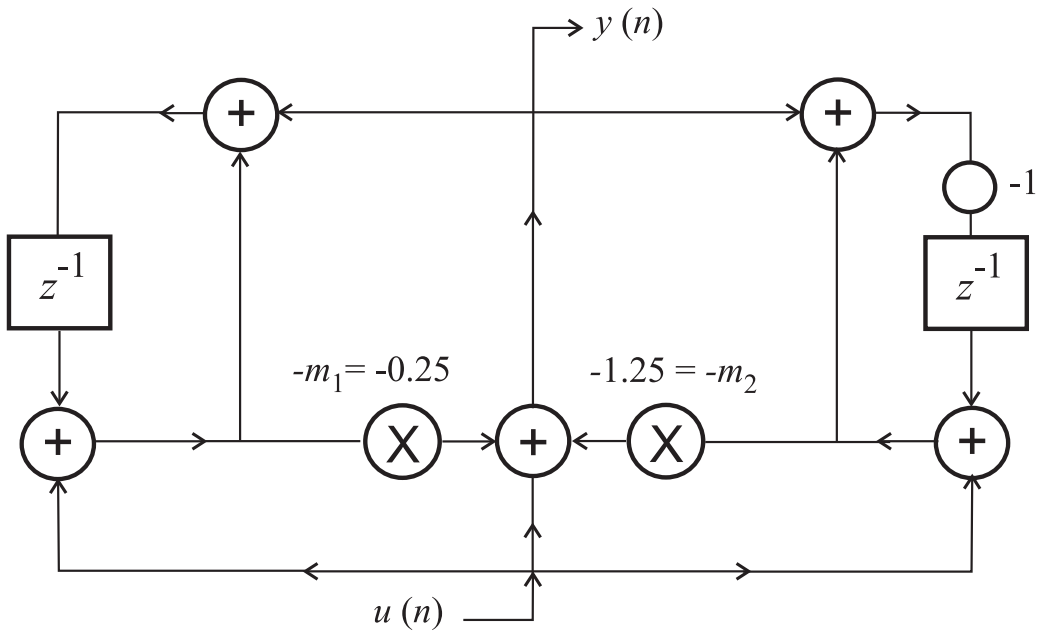


Figure 38: Digital filter structure.

Do-it-yourself: Finite-precision digital signal processing

- The corresponding transfer function is

$$H(z) = \begin{bmatrix} -m_1 & -m_2 \end{bmatrix} \begin{bmatrix} (z-1+m_1) & m_2 \\ -m_1 & (z-m_2+1) \end{bmatrix}^{-1} \begin{bmatrix} (2-m_1-m_2) \\ -(2-m_1-m_2) \end{bmatrix} + (1-m_1-m_2) \tag{222}$$

which, after some cumbersome algebraic development, becomes

$$H(z) = \frac{N(z)}{D(z)} = -\frac{(m_1+m_2-1)z^2 + (m_1-m_2)z + 1}{z^2 + (m_1-m_2)z + (m_1+m_2-1)} \tag{223}$$

corresponding to an all-pass second-order block.

Do-it-yourself: Finite-precision digital signal processing

- The transfer function from the filter input to the $-m_1$ multiplier is given by

$$F_1(z) = \frac{z^2 + 2(1 - m_2)z + 1}{z^2 + (m_1 - m_2)z + (m_1 + m_2 - 1)} \tag{224}$$

and the transfer function from the filter input to the $-m_2$ multiplier is

$$F_2(z) = \frac{z^2 + 2(m_1 - 1)z + 1}{z^2 + (m_1 - m_2)z + (m_1 + m_2 - 1)} \tag{225}$$

Do-it-yourself: Finite-precision digital signal processing

- Determining the L_2 norm for each scaling transfer function in a closed form is quite computationally intensive. Using MATLAB, however, this can be done numerically using few commands, such as:

```
m1 = 0.25; m2 = 1.25;
N1 = [1 2*(1-m2) 1]; N2 = [1 2*(m1-1) 1];
D = [1 (m1-m2) (m1+m2-1)];
np = 1000;
[F1,f] = freqz(N1,D,np); F1_2 =
sqrt((sum(abs(F1).^2))/np);
[F2,f] = freqz(N2,D,np); F2_2 =
sqrt((sum(abs(F2).^2))/np);
```

Do-it-yourself: Finite-precision digital signal processing

- As a result, F1_2 and F2_2 are equal to 1.7332 and 1.1830, respectively.
- Then, we should scale the filter input by a factor of

$$\lambda = \frac{1}{\max\limits_{i=1,2} [\|F_i(z)\|_2]} = \frac{1}{\max [1.7332, 1.1830]} = 0.5770 \qquad (226)$$

and compensate for this by multiplying the filter output by $g = \frac{1}{\lambda} = 1.7332$.

Do-it-yourself: Finite-precision digital signal processing

- The transfer functions from the outputs of both multipliers $-m_1$ and $-m_2$ to the filter output are expressed as

$$G_1(z) = G_2(z) = H(z) \tag{227}$$

such that

$$\|G_1(z)\|_2 = \|G_2(z)\|_2 = 1 \tag{228}$$

as $H(z)$ represents an all-pass filter with unit gain.

- Therefore, considering the scaling operation performed as above, the output noise variance is given by

$$\sigma_y^2 = 3g^2\sigma_e^2 + \sigma_e^2 = 10.0120\sigma_e^2 \tag{229}$$

where the factor 3 accounts for the noise sources in the input scaling, $-m_1$, and $-m_2$ multipliers.