SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL AND ELECTRONICS

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

UNIT – I - SIGNALS, SYSTEMS & TRANSFORMS– SEC1315

I. SIGNALS, SYSTEMS & TRANSFORMS

1.1 INTRODUCTION

A signal is a function of independent variables such as time, distance, position, temperature and pressure. A signal carries information, and the objective of signal processing is to extract useful information carried by the signal.

Signal processing is concerned with the mathematical representation of the signal and the algorithmic operation carried out on it to extract the information present. For most purposes of description and analysis, a signal can be defined simply as a mathematical function, y where x is the independent variable .y = f (x) .e.g.: y=sin(ωt) is a function of a variable in the time domain and is thus a time signal.X(ω)=1/(-mω^2+icω+k) is a frequency domain signal; An image I(x,y) is in the spatial domain.

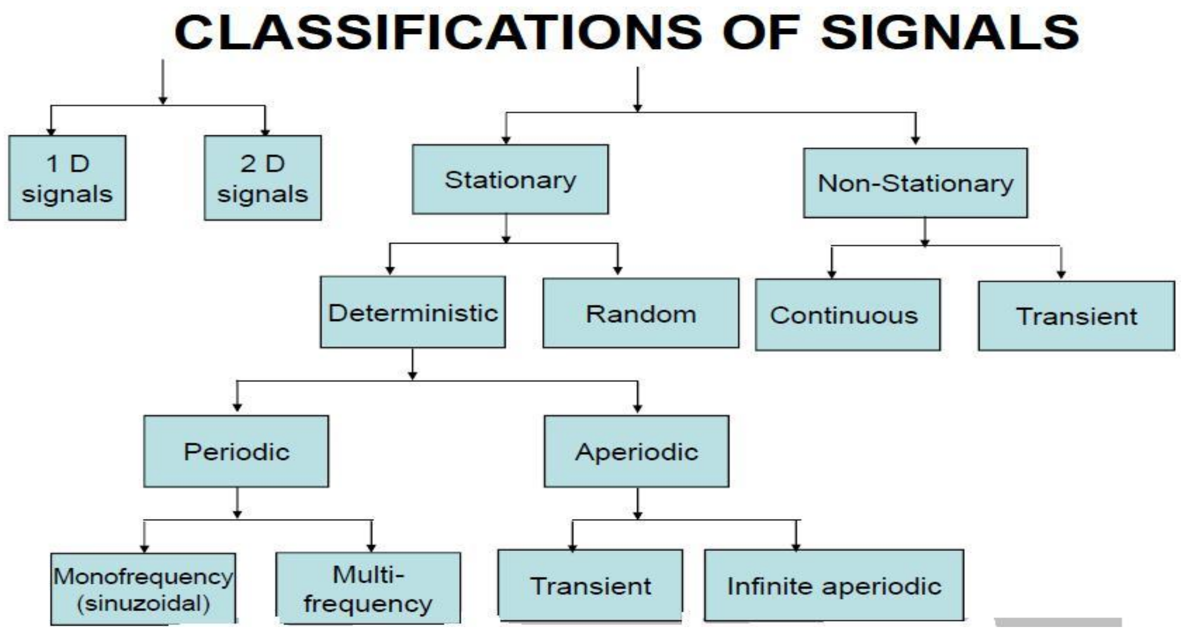


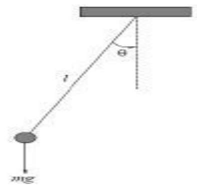
Fig 1.1 Classification of signals

At *t*=0, will have the same motions at all time. There is no place for uncertainty here. If we can uniquely specify the value of θ for all time, *i.e.*, we know the underlying functional relationship between *t* andθ, the motion is **deterministic** or predictable. In other words, a signal that can be uniquely determined by a well defined process such as a mathematical expression or rule is called a **deterministic signal**. The opposite situation occurs if we know all the physics there is to know, but still cannot say what the signal will be at the next time instant-then the signal is

random or **probabilistic.** In other words, a signal that is generated in a random fashion and can not be predicted ahead of time is called a random signal.

1.1.1 EXAMPLES OF SIGNALS

For a simple pendulum as shown, basic definition is: where θ*m* is the peak amplitude of the motion and ω=√*l/g* with *l* the length of the pendulum and *g* the acceleration due to gravity. As the system has a constant amplitude (we assume no damping for now), a constant frequency (dictated by physics) and an initial condition (θ=0 when *t*=0), we know the value of θ(*t*) for all time



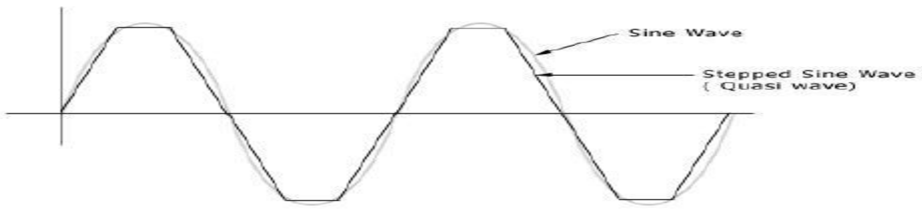


Fig 1.2 Typical examples to deterministic signals

Random signals are characterized by having many frequency components present over a wide range of frequencies.The amplitude versus time appears to vary rapidly and unsteadily with time. The 'shhhh' sound is a good example that is rather easy to observe using a microphone and oscillloscope. If the sound intensity is constant with time, the random signal is stationary, while if the sound intensity varies with time the signal is nonstationary. One can easily see and hear this variation while making the 'shhhh' sound.

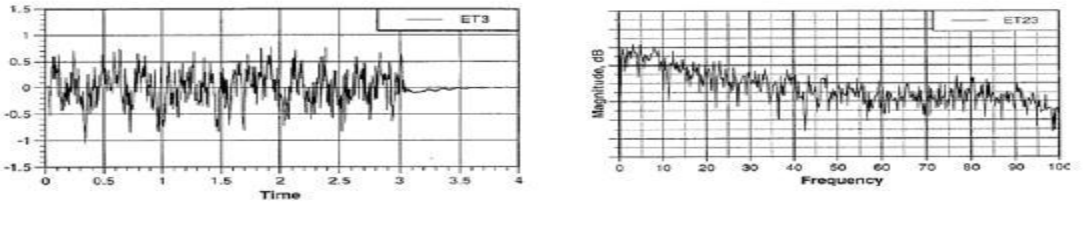


Fig 1.3 Random signals

Random signals are characterized by analyzing the statistical characteristics across an ensemble of records. Then, if the process is ergodic, the time (temporal) statistical characteristics are the same as the ensemble statistical characteristics. The word temporal means that a time average definition is used in place of an ensemble statistical definition

Transient signals may be defined as signals that exist for a finite range of time as shown in the figure. Typical examples are hammer excitation of systems, explosion and shock loading etc. It should be noted that periodicity does not necessarily mean a sinusoidal signal as shown in the figure.

$$S(t) = \sum_{i=1}^{\infty} s_i \sin\left(\frac{2\pi i}{\tau} . t\right)$$

A signal with a time varying mean is an **aperiodic** signal

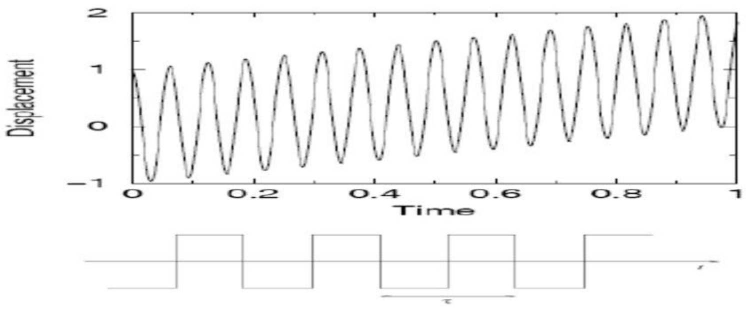


Fig 1.4 Aperiodic signal

For a simple pendulum as shown, if we define the period τ by , then for the pendulum, and such signals are defined as periodic. A periodic signal is one that repeats itself in time and is a reasonable model for many real processes, especially those associated with constant speed machinery. •**Stationary signals** are those whose average properties do not change with time. **Stationary signals** have constant parameters to describe their behaviour.**Nonstationary signals** have time dependent parameters. In an engine excited vibration where the engines speed varies with time; the fundamental period changes with time as well as with the corresponding dynamic loads that cause vibration.

Deterministic Vs Random Signal:

The signals can be further classified as **monofrequency** (sinusoidal) signals and **multifrequency** signals such as the square wave which has a functional form made up of an infinite superposition of different sine waves with periods τ,τ/2,τ/3,… .**1 D signals** are a function of a single independent variable. The speech signal is an example of a 1 D signal where the independent variable is time.. **2D signals** are a function of two independent variables. An image signal such as a photograph is an example of a 2D signal where the two independent variables are the two spatial variables

1.1.2 CONTINUOUS VERSUS DISCRETE SIGNALS

The value of a signal at a specific value of the independent variable is called its **amplitude**.The variation of the amplitude as a function of the independent variable is called its **waveform**.For a 1 D signal, the independent variable is usually labelled as time. If the independent variable is continuous, the signal is called a **continuous-time signal**. A continuous time signal is defined at every instant of time.If the independent variable is discrete, the signal is called a **discrete-time signal**. A discrete time signal takes certain numerical values at specified discrete instants of time, and between these specified instants of time, the signal is not defined. Hence, a discrete time signal is basically a sequence of numbers.

1.1.3 ANALOG VERSUS DIGITAL SIGNALS

A continuous-time signal with a continuous amplitude is usually called an **analog signal**. A speech signal is an example of an analog signal.

A discrete time signal with discrete valued amplitudes represented by a finite number of digits is referred to as a **digital signal**

1.2 SAMPLING AND QUANTIZATION

Nearly all data acquisition systems sample data with uniform time intervals. For evenly sampled data, time can be expressed as:

$T = (N\ 1)\ \ t$. where N is the sampling index which is the number of equally spaced samples. For most Fourier analyzers N is restricted to a power of 2.

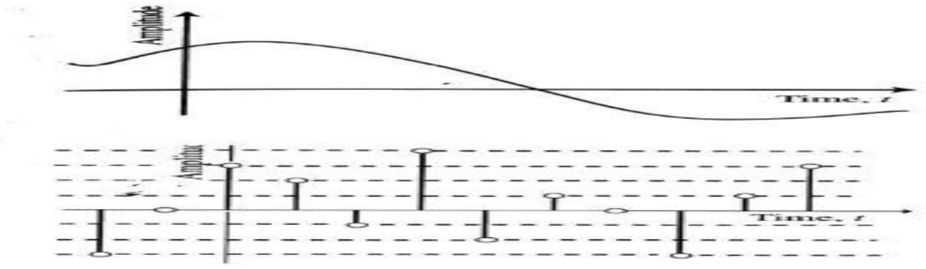


Fig 1. 5 Process of sampling

• The sample rate or the sampling frequency is:

$f = 1 = (N - 1)\ f$

Sampling frequency is the reciprocal of the time elapsed   t from one sample to the next.

• The unit of the sampling frequency is cycles per second or Hertz (Hz), if the sampling period is in seconds.

• The sampling theorem asserts that the uniformly spaced discrete samples are a complete representation of the signal if the bandwidth *fmax* is less than half the sampling rate. The sufficient condition for exact reconstructability from samples at a uniform sampling rate *fs* (in samples per unit time) ($fs \geq 2fmax$).

1.2.1 Aliasing

One problem encountered in A/D conversion is that a high frequency signal can be falsely confused as a low frequency signal when sufficient precautions have been avoided.This happens when the sample rate is not fast enough for the signal and one speaks of aliasing.Unfortunately, this problem can not always be resolved by just sampling faster, the signal's frequency content must also be limited. Furthermore, the costs involved with postprocessing and data analysis increase with the quantity of data obtained.

Data acquisition systems have finite memory, speed and data storage capabilities. Highly oversampling a signal can necessitate shorter sample lengths, longer time on test, more storage medium and increased database management and archiving requirements The central concept to avoid aliasing is that the sample rate must be at least twice the highest frequency component of the signal ($fs \geq 2fmax$).

We define the Nyquist or cut-off frequency.The concept behind the cut-off frequency is often referred to as 2 *t.* Shannon's sampling criterion. Signal components with frequency content above the cut-off frequency are aliased and can not be distinguished from the frequency components below the cut-off frequency.

Conversion of analog frequency into digital frequency during sampling is shown in the figure. Continuous signals with a frequency less **than** one-half of the sampling rate are directly converted into the corresponding digital frequency. Above one-half of the sampling rate, aliasing takes place, resulting in the frequency being misrepresented in the digital data. Aliasing always changes a higher frequency into a lower frequency between 0 and 0.5. In addition, aliasing may also change the phase of the signal by 180 degrees.

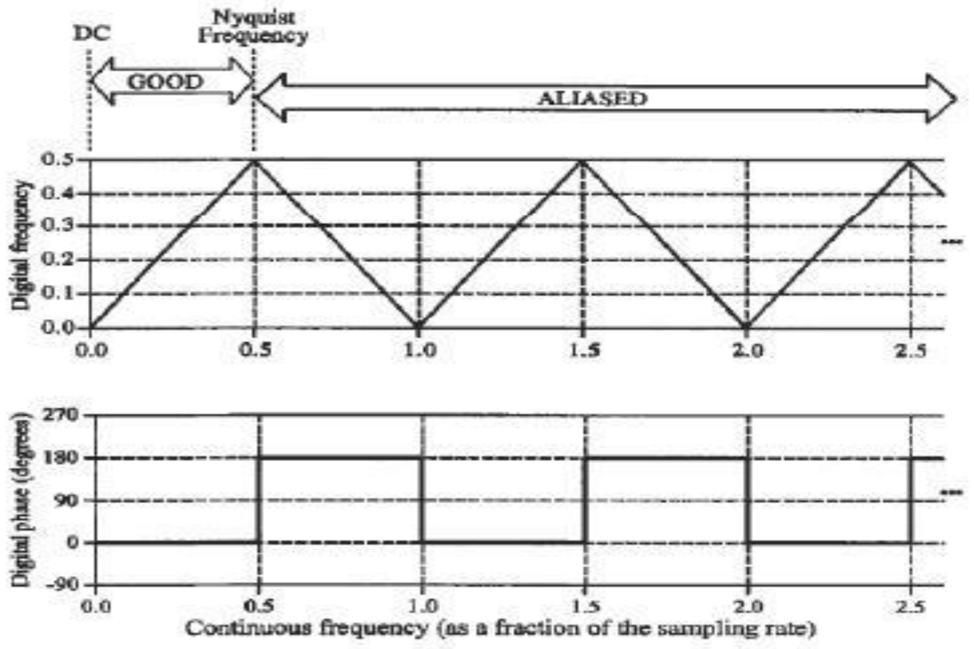
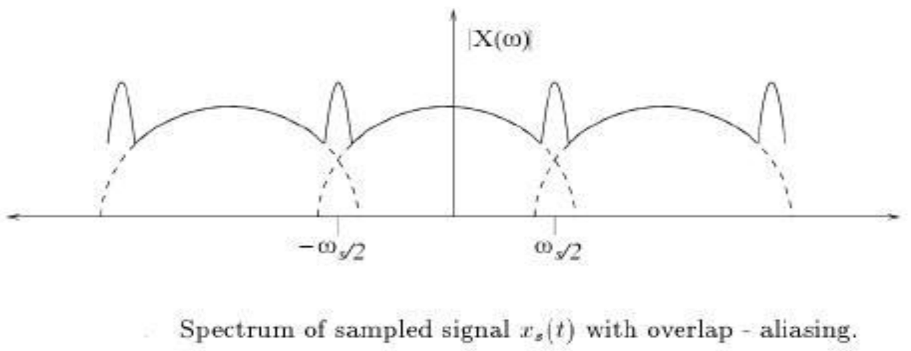


Fig 1.6 Aliasing effect

If any energy in the original signal extends beyond the Nyquist frequency, it is folded back into the Nyquist interval in the spectrum of the sampled signal. This folding is called aliasing.fs≥2fmax



|X(ω)|
$-\omega_{s/2}$
$\omega_{s/2}$
Spectrum of sampled signal $x_s(t)$ with overlap - aliasing.

Fig 1.7 Spectrum of sampled signal

QuantizationQuantization is involved to some degree in nearly all digital signal processing, as the process of representing a signal in digital form ordinarily involves rounding. Quantization also forms the core of essentially all lossy compression algorithms. The difference between an input value and its quantized value (such as round-off error) is referred to as **quantization error**. A device or algorithmic function that performs quantization is called a **quantizer**. An analog-to-digital converter is an example of a quantizer. Because quantization is a many-to-few mapping, it is an inherently non-linear and irreversible process (i.e., because the same output value is shared by multiple input values, it is impossible in general to recover the exact input value when given only the output value). The set of possible input values may be infinitely large, and may possibly be continuous and therefore uncountable (such as the set of all real numbers, or all real numbers within some limited range). The set of possible output values may be finite or countably infinite. The input and output sets involved in quantization can be defined in a rather general way. For example, *vector quantization* is the application of quantization to multi-dimensional (vector-valued) input data.

1.3 CONCEPTS OF SIGNAL PROCESSING

In the case of **analog signals**, most signal processing operations are usually carried out in the **time domain**.In the case of **discrete time signals**, **both time domain and frequency domain** applications are employed.In either case, the desired operations are implemented by a combination of some **elementary operations** such as:

– Simple time domain operations , Filtering , Amplitude modulation

The three most basic time-domain signal operations are:

• **Scaling**

• **Delay**

• **Addition**

Scaling is simply the multiplication of a signal by a positive or a negative constant. In the case of analog signals, this operation is usually called **amplification** if the magnitude of the multiplying constant, called **gain**, is greater than one. If the magnitude of the multiplying constant is less than one, the operation is called **attenuation.** Thus, if $x(t)$ is an analog signal, the scaling operation generates a signal $y(t)=\alpha x(t)$, where α is the multiplying constant.

Delay operation generates a signal that is delayed replica of the original signal. For an analog signal x(t), y(t)=x(t-t0) is the signal obtained by delaying x(t) by the amount t0, which is assumed to be a positive number. If t0 is negative, then it is an advance operation Addition operation generates a new signal by the addition of signals. For instance, y(t)=x1(t)+x2(t)-x3(t) is the signal generated by the addition of the three analog signals x1(t), x2(t) and x3(t) .

1.4 TYPICAL APPLICATIONS

The main applications of DSP are

AUDIO SIGNAL PROCESSING, sometimes referred to as audio processing, is the intentional alteration of auditory signals, or sound, often through an audio effect oreffects unit. As audio signals may be electronically represented in either digital or analog format, signal processing may occur in either domain. Analog processors operate directly on the electrical signal, while digital processors operate mathematically on the digital representation of that signal.

AUDIO COMPRESSION, bit-rate reduction involves encoding information using fewer bits than the original representation.[2]Compression can be either lossy or lossless. Lossless compression reduces bits by identifying and eliminating statistical redundancy. No information is lost in lossless compression. Lossy compression reduces bits by identifying unnecessary information and removing it.[3] The process of reducing the size of a data file is referred to as data compression. In the context of data transmission, it is called source coding (encoding done at the source of the data before it is stored or transmitted) in opposition to channel coding.[4]

DIGITAL IMAGE PROCESSING, is the use of computer algorithms to perform image processing on digital images. As a subcategory or field of digital signal processing, digital image processing has many advantages over analog image processing. It allows a much wider range of algorithms to be applied to the input data and can avoid problems such as the build-up of noise and signal distortion during processing. Since images are defined over two dimensions (perhaps more) digital image processing may be model in the form of multidimensional systems

SPEECH PROCESSING,s the study of speech signals and the processing methods of these signals. The signals are usually processed in a digital representation, so speech processing can be regarded as a special case of digital signal processing, applied to speech signal. Aspects of speech processing includes the acquisition, manipulation, storage, transfer and output of speech signals.

SPEECH RECOGNITION, is the inter-disciplinary sub-field of computational linguistics which incorporates knowledge and research in the linguistics, computer science, and electrical engineering fields to develop methodologies and technologies that enables the recognition and translation of spoken language into text by computers and computerized devices such as those categorized as Smart Technologies and robotics. It is also known as "automatic speech recognition" (ASR), "computer speech recognition", or just "speech to text" (STT). imaging such as CAT scans and MRI, MP3 compression, computer graphics, image manipulation, hi-fi loudspeakercrossovers and equalization, and audio effects for use with electric guitar amplifiers.

1.4.1 ADVANTAGES OF DIGITAL SIGNAL PROCESSING COMPARED WITH ANALOG SIGNAL PROCESSING

Accracy

Implimentation of sophisticated algorithms

Storage

Noise reduction

1.4.2 APPLICATIONS OF SIGNAL PROCESSING IN BIOMEDICAL ENGINEERING

I/0 signal processing – for electrical signals representing sound, such as speech or music ,

Speech signal processing – for processing and interpreting spoken words

Image processing – in digital cameras, computers and various imaging systems,

Video processing – for interpreting moving pictures,

Wireless communication - waveform generations, demodulation, filtering, equalization,Control systems,

Array processing – for processing signals from arrays of sensors, Seismology,

Financial signal processing – analyzing financial data using signal processing techniques, especially for prediction purposes.

Feature extraction, such as image understanding and ,

Quality improvement, such as noise reduction,

image enhancement, and echo cancellation.(Source coding), including audio compression, image compression, and video compression

1.5 DISCRETE TIME SIGNALS

A discrete time signal is defined as the one that is defined at distinct time intervals

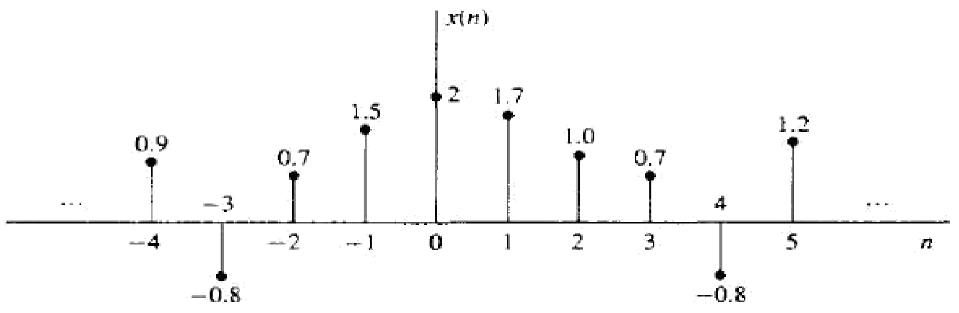


Fig 1.8 Graphical representation of a discrete-time signal.

two successive samples. Also, it is incorrect to think that $x(n)$ is equal to zero if n is not an integer. Simply, the signal $x(n)$ is not defined for noninteger values of n.

In the sequel we will assume that a discrete-time signal is defined for every integer value n for $-\infty < n < \infty$. By tradition, we refer to $x(n)$ as the "nth sample" of the signal even if the signal $x(n)$ is inherently discrete time (i.e., not obtained by sampling an analog signal). If, indeed, $x(n)$ was obtained from sampling an analog signal $x_a(t)$, then $x(n) \equiv x_a(nT)$, where T is the sampling period (i.e., the time between successive samples).

Besides the graphical representation of a discrete-time signal or sequence as illustrated in Fig. 2.1, there are some alternative representations that are often more convenient to use. These are:

1. Functional representation, such as

$$x(n) = \begin{cases} 1, & \text{for } n = 1, 3 \\ 4, & \text{for } n = 2 \\ 0, & \text{elsewhere} \end{cases}$$

2. Tabular representation, such as

n	$\cdots$	-2	-1	0	1	2	3	4	5	$\ldots$
$x(n)$	$\cdots$	0	0	0	1	4	1	0	0	$\ldots$

3. Sequence representation

An infinite-duration signal or sequence with the time origin ($n = 0$) indicated by the symbol $\uparrow$ is represented as

$$x(n) = \{\ldots 0, \underset{\uparrow}{0}, 1, 4, 1, 0, 0, \ldots\}$$

A sequence $x(n)$, which is zero for $n < 0$, can be represented as

$$x(n) = \{\underset{\uparrow}{0}, 1, 4, 1, 0, 0, \ldots\}$$

The time origin for a sequence $x(n)$, which is zero for $n < 0$, is understood to be the first (leftmost) point in the sequence.

A finite-duration sequence can be represented as

$$x(n) = \{3, -1, \underset{\uparrow}{-2}, 5, 0, 4, -1\}$$

whereas a finite-duration sequence that satisfies the condition $x(n) = 0$ for $n < 0$ can be represented as

$$x(n) = \{\underset{\uparrow}{0}, 1, 4, 1\}$$

The signal in (2.1.4) consists of seven samples or points (in time), so it is called or identified as a seven-point sequence. Similarly, the sequence given by (2.1.5) is a four-point sequence.

Some Elementary Discrete-Time Signals

In our study of discrete-time signals and systems there are a number of basic signals that appear often and play an important role. These signals are defined below.

1. The *unit sample sequence* is denoted as $\delta(n)$ and is defined as

$$\delta(n) \equiv \begin{cases} 1, & \text{for } n = 0 \\ 0, & \text{for } n \neq 0 \end{cases}$$

   In words, the unit sample sequence is a signal that is zero everywhere, except at $n = 0$ where its value is unity. This signal is sometimes referred to as a *unit impulse*. In contrast to the analog signal $\delta(t)$, which is also called a unit impulse and is defined to be zero everywhere except $t = 0$, and has unit area, the unit sample sequence is much less mathematically complicated. The graphical representation of $\delta(n)$ is shown in Fig. 2.2.

2. The *unit step signal* is denoted as $u(n)$ and is defined as

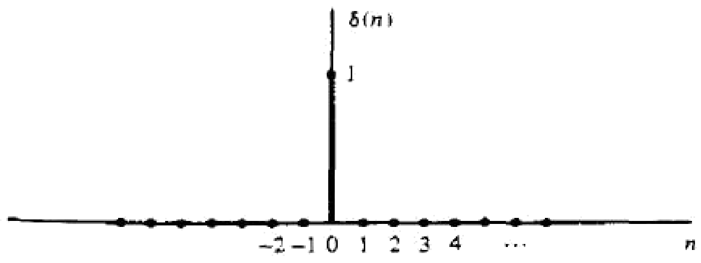
$$u(n) \equiv \begin{cases} 1, & \text{for } n \geq 0 \\ 0, & \text{for } n < 0 \end{cases}$$

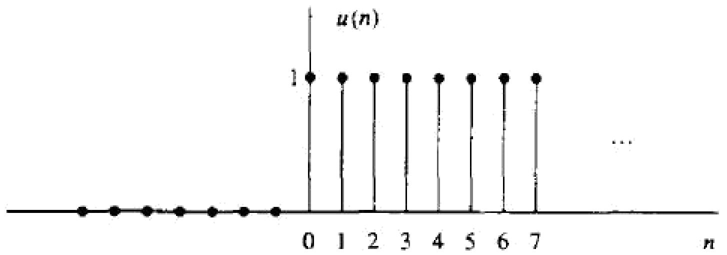
   Figure 2.3 illustrates the unit step signal.

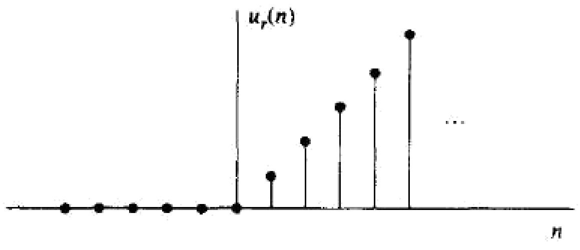
3. The *unit ramp signal* is denoted as $u_r(n)$ and is defined as

$$u_r(n) \equiv \begin{cases} n, & \text{for } n \geq 0 \\ 0, & \text{for } n < 0 \end{cases}$$

   This signal is illustrated in Fig. 2.4.







4. The *exponential signal* is a sequence of the form

$$x(n) = a^n \qquad \text{for all } n$$

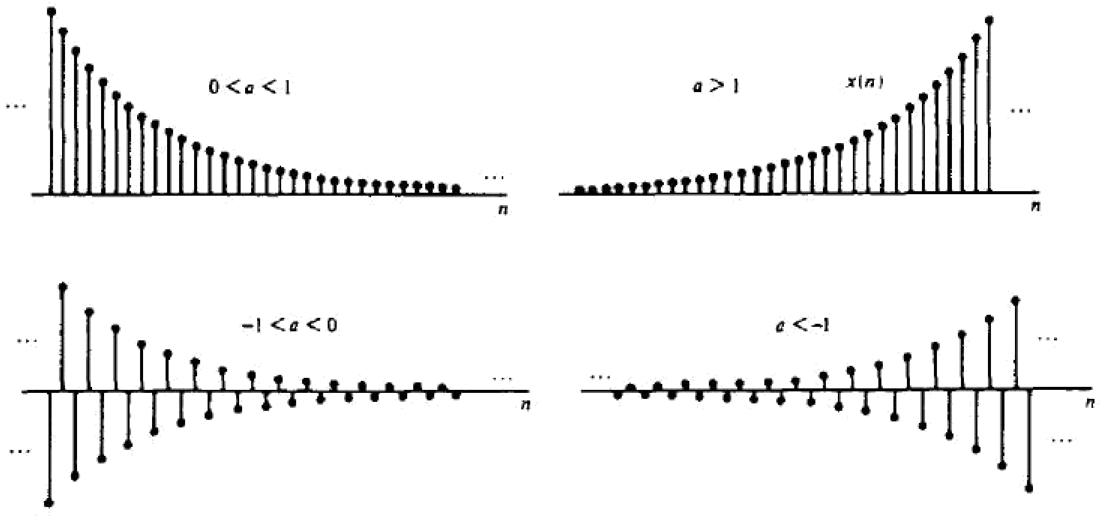
If the parameter a is real, then $x(n)$ is a real signal. Figure 2.5 illustrates $x(n)$ for various values of the parameter a.

When the parameter a is complex valued, it can be expressed as

$$a \equiv re^{j\theta}$$

where r and θ are now the parameters. Hence we can express $x(n)$ as

$$\begin{aligned} x(n) &= r^n e^{j\theta n} \\ &= r^n(\cos\theta n + j\sin\theta n) \end{aligned}$$



Graphical representation of exponential signals.

Since $x(n)$ is now complex valued, it can be represented graphically by plotting the real part

$$x_R(n) \equiv r^n \cos \theta n$$

as a function of n, and separately plotting the imaginary part

$$x_I(n) \equiv r^n \sin \theta n$$

as a function of n. Figure 2.6 illustrates the graphs of $x_R(n)$ and $x_I(n)$ for $r = 0.9$ and $\theta = \pi/10$. We observe that the signals $x_R(n)$ and $x_I(n)$ are a damped (decaying exponential) cosine function and a damped sine function. The angle variable θ is simply the frequency of the sinusoid, previously denoted by the (normalized) frequency variable ω. Clearly, if $r = 1$, the damping disappears and $x_R(n)$, $x_I(n)$, and $x(n)$ have a fixed amplitude, which is unity.

Alternatively, the signal $x(n)$ given by (2.1.10) can be represented graphically by the amplitude function

$$|x(n)| = A(n) \equiv r^n$$

and the phase function

$$\angle x(n) = \phi(n) \equiv \theta n$$

Figure 2.7 illustrates $A(n)$ and $\phi(n)$ for $r = 0.9$ and $\theta = \pi/10$. We observe that the phase function is linear with n. However, the phase is defined only over the interval $-\pi < \theta \leq \pi$ or, equivalently, over the interval $0 \leq \theta < 2\pi$. Consequently, by convention $\phi(n)$ is plotted over the finite interval $-\pi < \theta \leq \pi$ or $0 \leq \theta < 2\pi$. In other words, we subtract multiples of 2π from $\phi(n)$ before plotting. In one case, $\phi(n)$ is constrained to the range $-\pi < \theta \leq \pi$ and in the other case $\phi(n)$ is constrained to the range $0 \leq \theta < 2\pi$. The subtraction of multiples of 2π from $\phi(n)$ is equivalent to interpreting the function $\phi(n)$ as $\phi(n)$, modulo 2π. The graph for $\phi(n)$, modulo 2π, is shown in Fig. 2.7b.

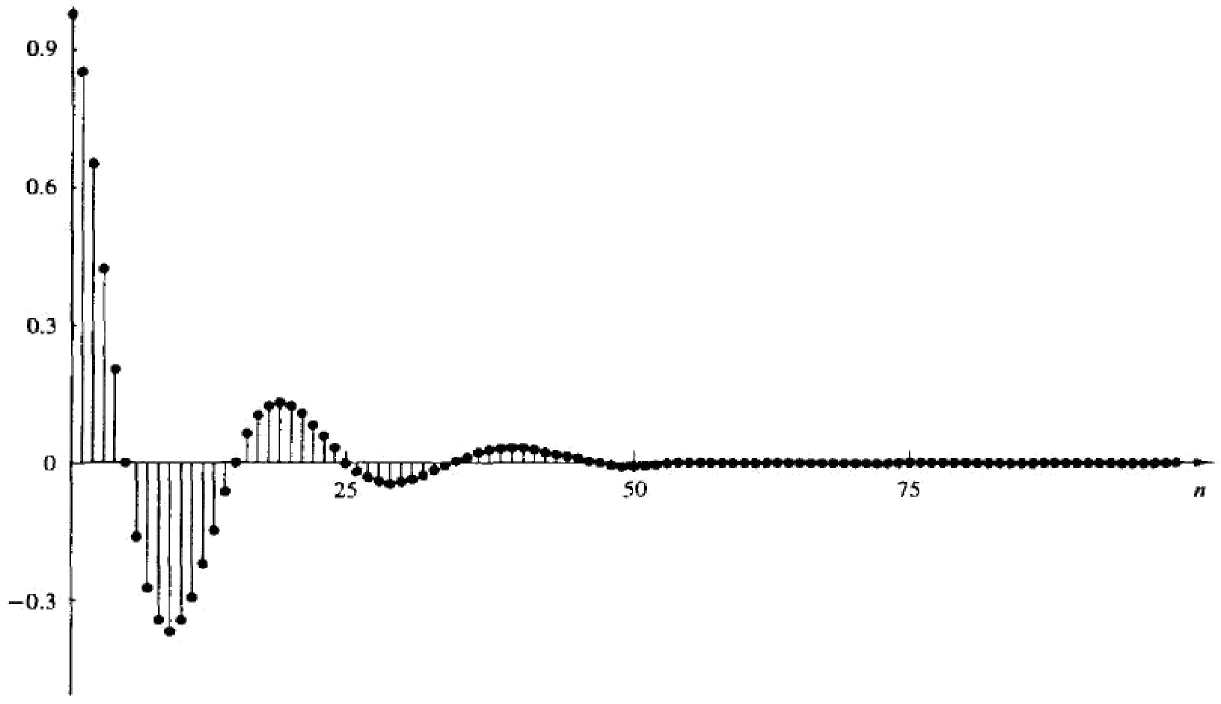
Classification of Discrete-Time Signals

The mathematical methods employed in the analysis of discrete-time signals and systems depend on the characteristics of the signals. In this section we classify discrete-time signals according to a number of different characteristics.

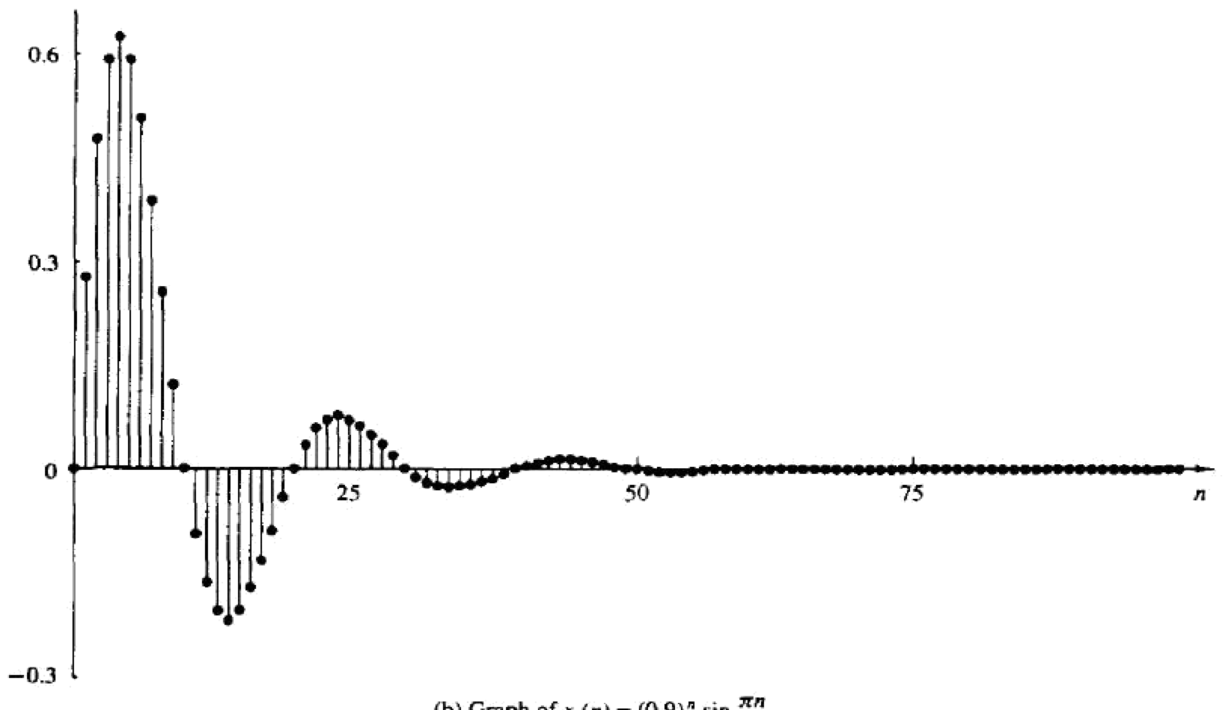
Energy signals and power signals. The energy E of a signal $x(n)$ is defined as

$$E \equiv \sum_{n=-\infty}^{\infty} |x(n)|^2$$

We have used the magnitude-squared values of $x(n)$, so that our definition applies to complex-valued signals as well as real-valued signals. The energy of a signal can be finite or infinite. If E is finite (i.e., $0 < E < \infty$), then $x(n)$ is called an *energy*

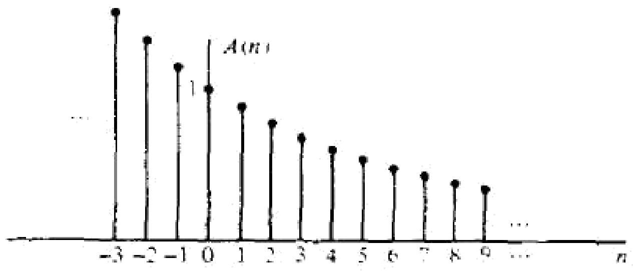


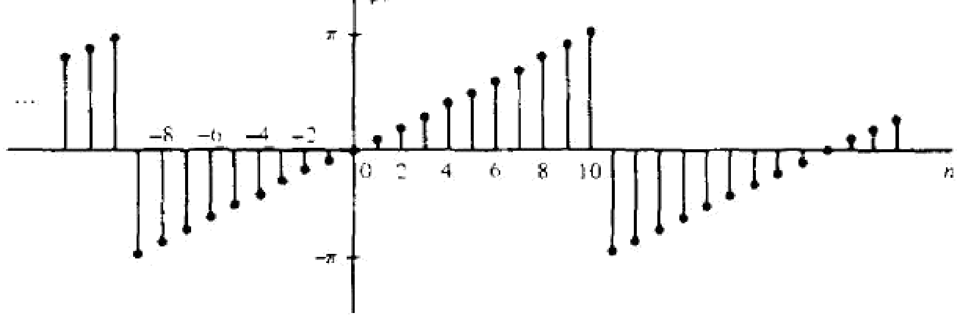
(a) Graph of $x_R(n) = (0.9)^n \cos \frac{\pi n}{10}$



(b) Graph of $x_I(n) = (0.9)^n \sin \frac{\pi n}{10}$

Graph of the real and imaginary components of a complex-valued exponential signal.


(a) Graph of $A(n) = r^n$, $r = 0.9$


(b) Graph of $\phi(n) = \frac{\pi}{10}n$, modulo 2π plotted in the range $(-\pi, \pi]$

Graph of amplitude and phase function of a complex-valued exponential signal: (a) graph of $A(n) = r^n$, $4 = 0.9$; (b) graph of $\phi(n) = (\pi/10)n$, modulo 2π plotted in the range $(-\pi, \pi]$.

signal. Sometimes we add a subscript x to E and write E_x to emphasize that E_x is the energy of the signal $x(n)$.

Many signals that possess infinite energy, have a finite average power. The average power of a discrete-time signal $x(n)$ is defined as

$$P = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} |x(n)|^2$$

If we define the signal energy of $x(n)$ over the finite interval $-N \le n \le N$ as

$$E_N \equiv \sum_{n=-N}^{N} |x(n)|^2$$

then we can express the signal energy E as

$$E \equiv \lim_{N\to\infty} E_N$$

and the average power of the signal $x(n)$ as

$$P \equiv \lim_{N\to\infty} \frac{1}{2N+1} E_N$$

Clearly, if E is finite, $P = 0$. On the other hand, if E is infinite, the average power P may be either finite or infinite. If P is finite (and nonzero), the signal is called a *power signal*. The following example illustrates such a signal.

Example 2.1.1

Determine the power and energy of the unit step sequence. The average power of the unit step signal is

$$\begin{aligned} P &= \lim_{N \to \infty} \frac{1}{2N+1} \sum_{n=0}^{N} u^2(n) \\ &= \lim_{N \to \infty} \frac{N+1}{2N+1} = \lim_{N \to \infty} \frac{1+1/N}{2+1/N} = \frac{1}{2} \end{aligned}$$

Consequently, the unit step sequence is a power signal. Its energy is infinite.

Similarly, it can be shown that the complex exponential sequence $x(n) = Ae^{j\omega_0 n}$ has average power A^2, so it is a power signal. On the other hand, the unit ramp sequence is neither a power signal nor an energy signal.

Periodic signals and aperiodic signals. As defined on Section 1.3, a signal $x(n)$ is periodic with period $N (N > 0)$ if and only if

$$x(n + N) = x(n) \text{ for all } n$$

The smallest value of N for which (2.1.20) holds is called the (fundamental) period. If there is no value of N that satisfies (2.1.20), the signal is called *nonperiodic* or *aperiodic*.

We have already observed that the sinusoidal signal of the form

$$x(n) = A \sin 2\pi f_0 n$$

is periodic when f_0 is a rational number, that is, if f_0 can be expressed as

$$f_0 = \frac{k}{N}$$

where k and N are integers.

The energy of a periodic signal $x(n)$ over a single period, say, over the interval $0 \le n \le N-1$, is finite if $x(n)$ takes on finite values over the period. However, the energy of the periodic signal for $-\infty \le n \le \infty$ is infinite. On the other hand, the average power of the periodic signal is finite and it is equal to the average power over a single period. Thus if $x(n)$ is a periodic signal with fundamental period N and takes on finite values, its power is given by

$$P = \frac{1}{N} \sum_{n=0}^{N-1} |x(n)|^2$$

Consequently, periodic signals are power signals.

Symmetric (even) and antisymmetric (odd) signals. A real-valued signal $x(n)$ is called symmetric (even) if

$$x(-n) = x(n)$$

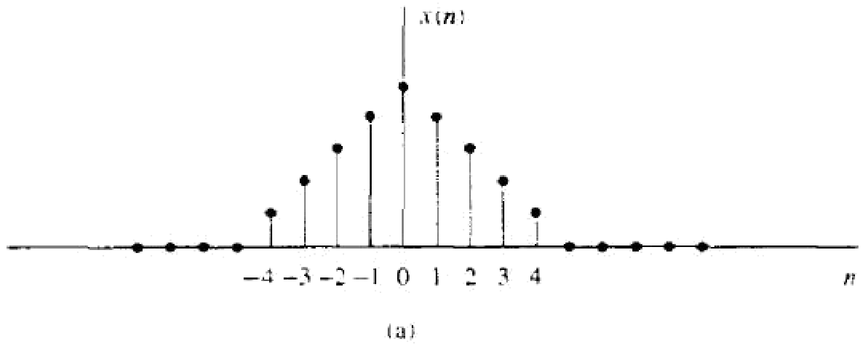
On the other hand, a signal $x(n)$ is called antisymmetric (odd) if

$$x(-n) = -x(n)$$

We note that if $x(n)$ is odd, then $x(0) = 0$. Examples of signals with even and odd symmetry are illustrated in Fig. 2.8.

We wish to illustrate that any arbitrary signal can be expressed as the sum of two signal components, one of which is even and the other odd. The even signal component is formed by adding $x(n)$ to $x(-n)$ and dividing by 2, that is,

$$x_e(n) = \tfrac{1}{2}[x(n) + x(-n)]$$



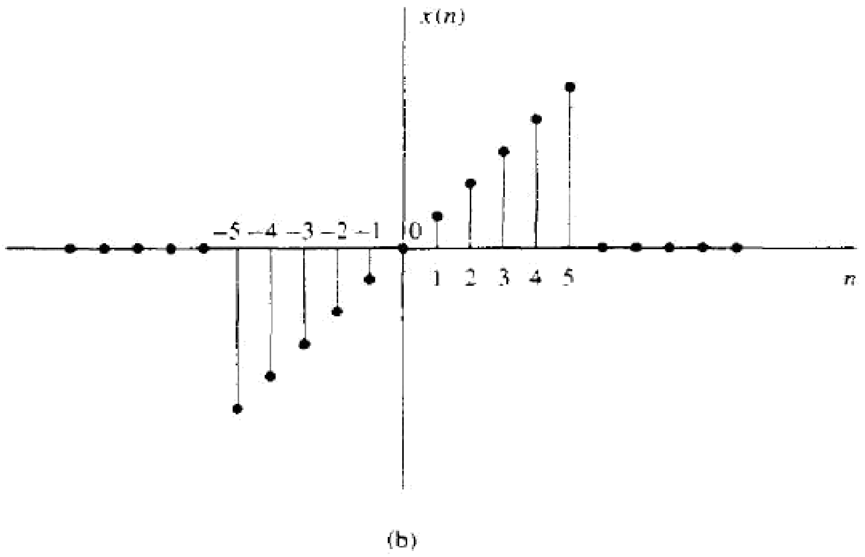


Figure 2.8 Example of even (a) and odd (b) signals.

Clearly, $x_e(n)$ satisfies the symmetry condition (2.1.24). Similarly, we form an odd signal component $x_o(n)$ according to the relation

$$x_o(n) = \frac{1}{2}[x(n) - x(-n)]$$

Again, it is clear that $x_o(n)$ satisfies (2.1.25); hence it is indeed odd. Now, if we add the two signal components, defined by (2.1.26) and (2.1.27), we obtain $x(n)$, that is,

$$x(n) = x_e(n) + x_o(n)$$

Thus any arbitrary signal can be expressed as in (2.1.28).

2.1.3 Simple Manipulations of Discrete-Time Signals

In this section we consider some simple modifications or manipulations involving the independent variable and the signal amplitude (dependent variable).

Transformation of the independent variable (time). A signal $x(n)$ may be shifted in time by replacing the independent variable n by $n - k$, where k is an integer. If k is a positive integer, the time shift results in a delay of the signal by k units of time. If k is a negative integer, the time shift results in an advance of the signal by $|k|$ units in time.

Example 2.1.2

A signal $x(n)$ is graphically illustrated in Fig. 2.9a. Show a graphical representation of the signals $x(n-3)$ and $x(n+2)$.

Solution The signal $x(n-3)$ is obtained by delaying $x(n)$ by three units in time. The result is illustrated in Fig. 2.9b. On the other hand, the signal $x(n+2)$ is obtained by advancing $x(n)$ by two units in time. The result is illustrated in Fig. 2.9c. Note that delay corresponds to shifting a signal to the right, whereas advance implies shifting the signal to the left on the time axis.

If the signal $x(n)$ is stored on magnetic tape or on a disk or, perhaps, in the memory of a computer, it is a relatively simple operation to modify the base by introducing a delay or an advance. On the other hand, if the signal is not stored but is being generated by some physical phenomenon in real time, it is not possible to advance the signal in time, since such an operation involves signal samples that have not yet been generated. Whereas it is always possible to insert a delay into signal samples that have already been generated, it is physically impossible to view the future signal samples. Consequently, in real-time signal processing applications, the operation of advancing the time base of the signal is physically unrealizable.

Another useful modification of the time base is to replace the independent variable n by $-n$. The result of this operation is a *folding* or a *reflection* of the signal about the time origin $n = 0$.

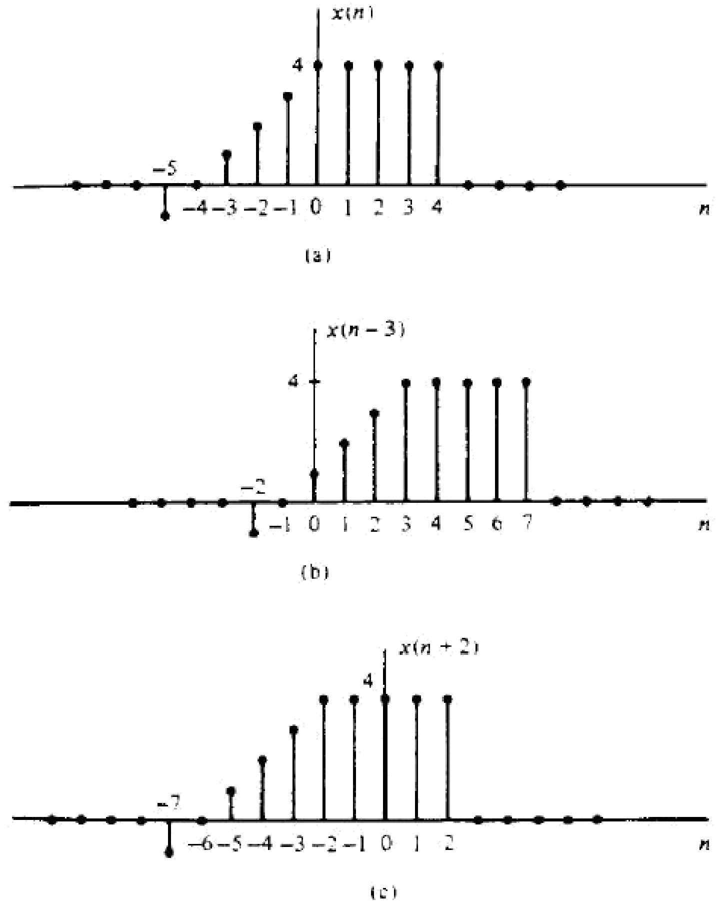


Figure 2.9 Graphical representation of a signal, and its delayed and advanced versions.

Example 2.1.3

Show the graphical representation of the signal $x(-n)$ and $x(-n+2)$, where $x(n)$ is the signal illustrated in Fig. 2.10a.

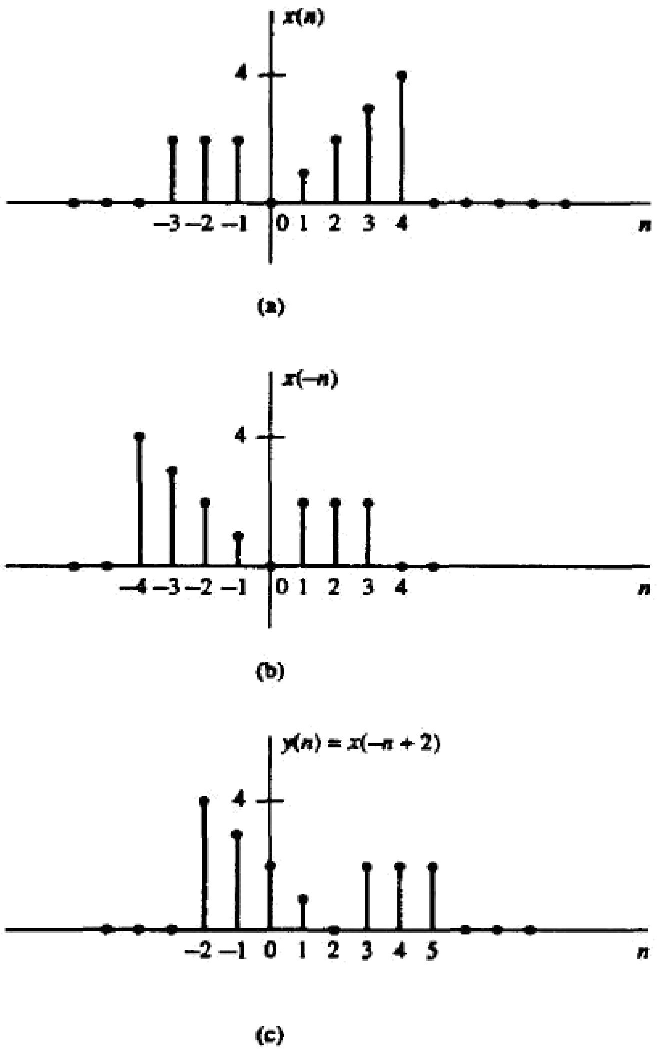
Solution The new signal $y(n) = x(-n)$ is shown in Fig. 2.10b. Note that $y(0) = x(0)$, $y(1) = x(-1)$, $y(2) = x(-2)$, and so on. Also, $y(-1) = x(1)$, $y(-2) = x(2)$, and so on. Therefore, $y(n)$ is simply $x(n)$ reflected or folded about the time origin $n = 0$. The signal $y(n) = x(-n+2)$ is simply $x(-n)$ delayed by two units in time. The resulting signal is illustrated in Fig. 2.10c. A simple way to verify that the result in Fig. 2.10c is correct is to compute samples, such as $y(0) = x(2)$, $y(1) = x(1)$, $y(2) = x(0)$, $y(-1) = x(3)$, and so on.

It is important to note that the operations of folding and time delaying (or advancing) a signal are not commutative. If we denote the time-delay operation by TD and the folding operation by FD, we can write

$$\begin{aligned} \mathrm{TD}_k[x(n)] &= x(n-k) \qquad k > 0 \\ \mathrm{FD}[x(n)] &= x(-n) \end{aligned} \tag{2.1.29}$$

Now

$$\mathrm{TD}_k\{\mathrm{FD}[x(n)]\} = \mathrm{TD}_k[x(-n)] = x(-n+k) \tag{2.1.30}$$



[illegible] Graphical illustration of the folding and shifting operations.

whereas

$$\mathrm{FD}\{\mathrm{TD}_k[x(n)]\} = \mathrm{FD}[x(n-k)] = x(-n-k)$$

Note that because the signs of n and k in $x(n-k)$ and $x(-n+k)$ are different, the result is a shift of the signals $x(n)$ and $x(-n)$ to the right by k samples, corresponding to a time delay.

A third modification of the independent variable involves replacing n by μn, where μ is an integer. We refer to this time-base modification as *time scaling* or *down-sampling*.

Example 2.1.4

Show the graphical representation of the signal $y(n) = x(2n)$, where $x(n)$ is the signal illustrated in Fig. 2.11a.

Solution We note that the signal $y(n)$ is obtained from $x(n)$ by taking every other sample from $x(n)$, starting with $x(0)$. Thus $y(0) = x(0)$, $y(1) = x(2)$, $y(2) = x(4)$, ... and $y(-1) = x(-2)$, $y(-2) = x(-4)$, and so on. In other words, we have skipped

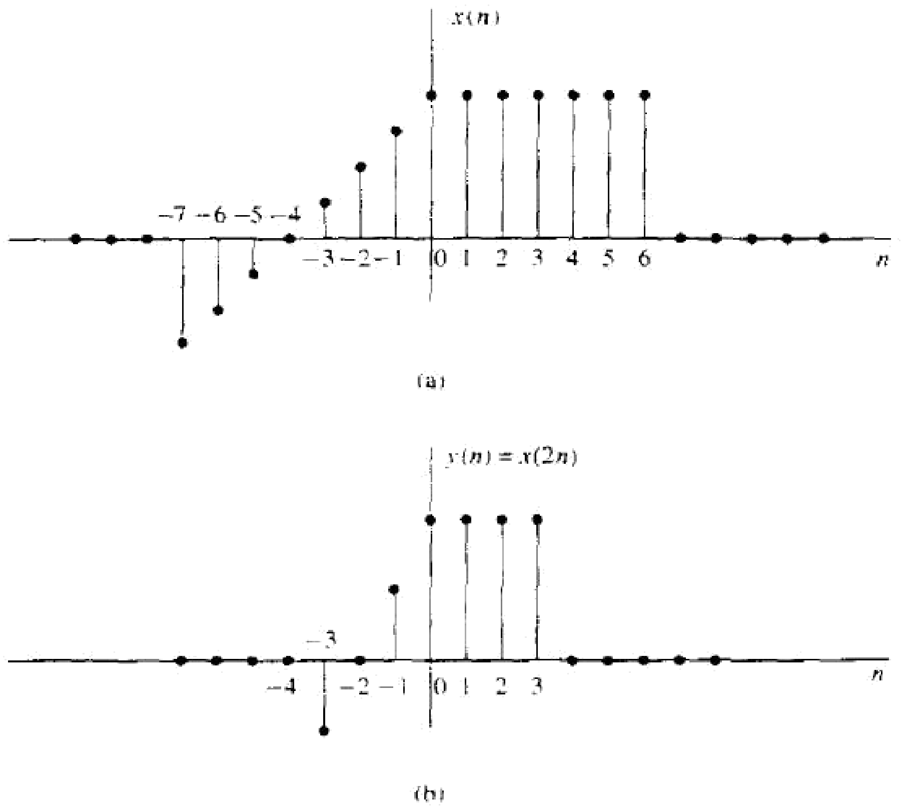


Figure 2.11 Graphical illustration of down-sampling operation.

the odd-numbered samples in $x(n)$ and retained the even-numbered samples. The resulting signal is illustrated in Fig. 2.11b.

If the signal $x(n)$ was originally obtained by sampling an analog signal $x_a(t)$, then $x(n) = x_a(nT)$, where T is the sampling interval. Now, $y(n) = x(2n) = x_a(2Tn)$. Hence the time-scaling operation described in Example 2.1.4 is equivalent to changing the sampling rate from $1/T$ to $1/2T$, that is, to decreasing the rate by a factor of 2. This is a *downsampling* operation.

Addition, multiplication, and scaling of sequences. Amplitude modifications include *addition, multiplication,* and *scaling* of discrete-time signals.

Amplitude scaling of a signal by a constant A is accomplished by multiplying the value of every signal sample by A. Consequently, we obtain

$$y(n) = Ax(n) \qquad -\infty < n < \infty$$

The *sum* of two signals $x_1(n)$ and $x_2(n)$ is a signal $y(n)$, whose value at any instant is equal to the sum of the values of these two signals at that instant, that is,

$$y(n) = x_1(n) + x_2(n) \qquad -\infty < n < \infty$$

The *product* of two signals is similarly defined on a sample-to-sample basis as

$$y(n) = x_1(n)x_2(n) \qquad -\infty < n < \infty$$

DISCRETE-TIME SYSTEMS

In many applications of digital signal processing we wish to design a device or an algorithm that performs some prescribed operation on a discrete-time signal. Such a device or algorithm is called a discrete-time system. More specifically, a *discrete-time system* is a device or algorithm that operates on a discrete-time signal, called the *input* or *excitation*, according to some well-defined rule, to produce another discrete-time signal called the *output* or *response* of the system. In general, we view a system as an operation or a set of operations performed on the input signal $x(n)$ to produce the output signal $y(n)$. We say that the input signal $x(n)$ is *transformed* by the system into a signal $y(n)$, and express the general relationship between $x(n)$ and $y(n)$ as

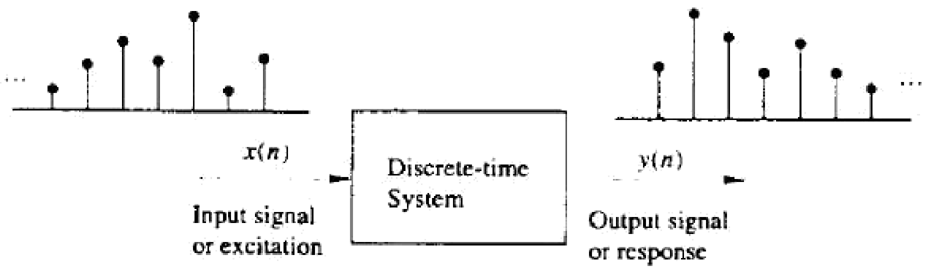
$$y(n) \equiv \mathcal{T}[x(n)]$$

where the symbol $\mathcal{T}$ denotes the transformation (also called an operator), or processing performed by the system on $x(n)$ to produce $y(n)$. The mathematical relationship in (2.2.1) is depicted graphically in Fig. 2.12.

There are various ways to describe the characteristics of the system and the operation it performs on $x(n)$ to produce $y(n)$. In this chapter we shall be concerned with the time-domain characterization of systems. We shall begin with an input–output description of the system. The input–output description focuses on the behavior at the terminals of the system and ignores the detailed internal construction or realization of the system. Later, in Section 7.5, we introduce the state-space description of a system. In this description we develop mathematical equations that not only describe the input–output behavior of the system but specify its internal behavior and structure.

Input–Output Description of Systems

The input–output description of a discrete-time system consists of a mathematical expression or a rule, which explicitly defines the relation between the input and output signals (*input–output relationship*). The exact internal structure of the system is either unknown or ignored. Thus the only way to interact with the system is by using its input and output terminals (i.e., the system is assumed to be a "black box" to the user). To reflect this philosophy, we use the graphical representa-



Block diagram representation of a discrete-time system.

tion depicted in Fig. 2.12, and the general input–output relationship in (2.2.1) or, alternatively, the notation

$$x(n) \xrightarrow{\mathcal{T}} y(n)$$

which simply means that $y(n)$ is the response of the system $\mathcal{T}$ to the excitation $x(n)$. The following examples illustrate several different systems.

Example 2.2.1

Determine the response of the following sytems to the input signal

$$x(n) = \begin{cases} |n|, & -3 \le n \le 3 \\ 0, & \text{otherwise} \end{cases}$$

(a) $y(n) = x(n)$

(b) $y(n) = x(n-1)$

(c) $y(n) = x(n+1)$

(d) $y(n) = \frac{1}{3}[x(n+1) + x(n) + x(n-1)]$

(e) $y(n) = max\{x(n+1), x(n), x(n-1)\}$

(f) $y(n) = \sum_{k=-\infty}^{n} x(k) = x(n) + x(n-1) + x(n-2) + \cdots$

Solution First, we determine explicitly the sample values of the input signal

$$x(n) = \{\ldots, 0, 3, 2, 1, \underset{\uparrow}{0}, 1, 2, 3, 0, \ldots\}$$

Next, we determine the output of each system using its input–output relationship.

(a) In this case the output is exactly the same as the input signal. Such a system is known as the *identity* system.

(b) This system simply delays the input by one sample. Thus its output is given by

$$x(n) = \{\ldots, 0, 3, 2, \underset{\uparrow}{1}, 0, 1, 2, 3, 0, \ldots\}$$

(c) In this case the system "advances" the input one sample into the future. For example, the value of the output at time $n = 0$ is $y(0) = x(1)$. The response of this system to the given input is

$$x(n) = \{\ldots, 0, 3, 2, 1, 0, \underset{\uparrow}{1}, 2, 3, 0 \ldots\}$$

(d) The output of this system at any time is the mean value of the present, the immediate past, and the immediate future samples. For example, the output at time $n = 0$ is

$$y(0) = \tfrac{1}{3}[x(-1) + x(0) + x(1)] = \tfrac{1}{3}[1 + 0 + 1] = \tfrac{2}{3}$$

Repeating this computation for every value of n, we obtain the output signal

$$y(n) = \{\ldots, 0, 1, \tfrac{5}{3}, 2, 1, \underset{\uparrow}{\tfrac{2}{3}}, 1, 2, \tfrac{5}{3}, 1, 0, \ldots\}$$

(e) This system selects as its output at time n the maximum value of the three input samples $x(n-1)$, $x(n)$, and $x(n+1)$. Thus the response of this system to the input signal $x(n)$ is

$$y(n) = \{0, 3, 3, 3, 2, \underset{\uparrow}{1}, 2, 3, 3, 3, 0, \ldots\}$$

(f) This system is basically an *accumulator* that computes the running sum of all the past input values up to present time. The response of this system to the given input is

$$y(n) = \{\ldots, 0, 3, 5, 6, \underset{\uparrow}{6}, 7, 9, 12, 0, \ldots\}$$

We observe that for several of the systems considered in Example 2.2.1 the tput at time $n = n_0$ depends not only on the value of the input at $n = n_0$ [i.e., $_0)$], but also on the values of the input applied to the system before and after $= n_0$. Consider, for instance, the accumulator in the example. We see that the tput at time $n = n_0$ depends not only on the input at time $n = n_0$, but also on $)$ at times $n = n_0 - 1$, $n_0 - 2$, and so on. By a simple algebraic manipulation : input–output relation of the accumulator can be written as

$$\begin{aligned} y(n) &= \sum_{k=-\infty}^{n} x(k) = \sum_{k=-\infty}^{n-1} x(k) + x(n) \\ &= y(n-1) + x(n) \end{aligned}$$

ich justifies the term *accumulator*. Indeed, the system computes the current ue of the output by adding (accumulating) the current value of the input to the :vious output value.

There are some interesting conclusions that can be drawn by taking a close ›k into this apparently simple system. Suppose that we are given the input signal $)$ for $n \geq n_0$, and we wish to determine the output $y(n)$ of this system for $n \geq n_0$. r $n = n_0, n_0 + 1, \ldots$, (2.2.4) gives

$$\begin{aligned} y(n_0) &= y(n_0 - 1) + x(n_0) \\ y(n_0 + 1) &= y(n_0) + x(n_0 + 1) \end{aligned}$$

1 so on. Note that we have a problem in computing $y(n_0)$, since it depends on $_0 - 1)$. However,

$$y(n_0 - 1) = \sum_{k=-\infty}^{n_0 - 1} x(k)$$

ıt is, $y(n_0 - 1)$ "summarizes" the effect on the system from all the inputs which 1 been applied to the system before time n_0. Thus the response of the system $n \geq n_0$ to the input $x(n)$ that is applied at time n_0 is the combined result of this ›ut and all inputs that had been applied previously to the system. Consequently, $)$, $n \geq n_0$ is not uniquely determined by the input $x(n)$ for $n \geq n_0$.

The additional information required to determine $y(n)$ for $n \geq n_0$ is the ***initial condition*** $y(n_0 - 1)$. This value summarizes the effect of all previous inputs to the system. Thus the initial condition $y(n_0 - 1)$ together with the input sequence $x(n)$ for $n \geq n_0$ uniquely determine the output sequence $y(n)$ for $n \geq n_0$.

If the accumulator had no excitation prior to n_0, the initial condition is $y(n_0 - 1) = 0$. In such a case we say that the system is *initially relaxed*. Since $y(n_0-1) = 0$, the output sequence $y(n)$ depends only on the input sequence $x(n)$ for $n \geq n_0$.

It is customary to assume that every system is relaxed at $n = -\infty$. In this case, if an input $x(n)$ is applied at $n = -\infty$, the corresponding output $y(n)$ is *solely* and *uniquely* determined by the given input.

Example 2.2.2

The accumulator described by (2.2.3) is excited by the sequence $x(n) = nu(n)$. Determine its output under the condition that:

(a) It is initially relaxed [i.e., $y(-1) = 0$].
(b) Initially, $y(-1) = 1$.

Solution The output of the system is defined as

$$y(n) = \sum_{k=-\infty}^{n} x(k) = \sum_{k=-\infty}^{-1} x(k) + \sum_{k=0}^{n} x(k)$$

$$= y(-1) + \sum_{k=0}^{n} x(k)$$

But

$$\sum_{k=0}^{n} x(k) = \frac{n(n+1)}{2}$$

(a) If the system is initially relaxed, $y(-1) = 0$ and hence

$$y(n) = \frac{n(n+1)}{2} \qquad n \geq 0$$

(b) On the other hand, if the initial condition is $y(-1) = 1$, then

$$y(n) = 1 + \frac{n(n+1)}{2} = \frac{n^2+n+2}{2} \qquad n \geq 0$$

2.2.2 Block Diagram Representation of Discrete-Time Systems

It is useful at this point to introduce a block diagram representation of discrete-time systems. For this purpose we need to define some basic building blocks that can be interconnected to form complex systems.

An adder. Figure 2.13 illustrates a system (adder) that performs the addition of two signal sequences to form another (the sum) sequence, which we denote

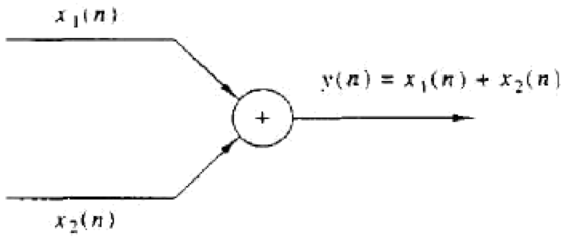


Figure 2.13 Graphical representation of an adder.

as $y(n)$. Note that it is not necessary to store either one of the sequences in order to perform the addition. In other words, the addition operation is *memoryless.*

A constant multiplier. This operation is depicted by Fig. 2.14, and simply represents applying a scale factor on the input $x(n)$. Note that this operation is also memoryless.

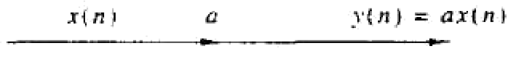
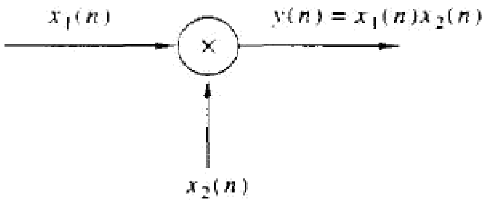


Figure 2.14 Graphical representation of a constant multiplier.

A signal multiplier. Figure 2.15 illustrates the multiplication of two signal sequences to form another (the product) sequence, denoted in the figure as $y(n)$. As in the preceding two cases, we can view the multiplication operation as memoryless.

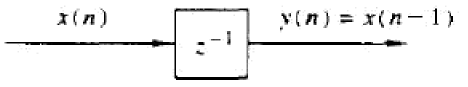


Graphical representation of a signal multiplier.

A unit delay element. The unit delay is a special system that simply delays the signal passing through it by one sample. Figure 2.16 illustrates such a system. If the input signal is $x(n)$, the output is $x(n-1)$. In fact, the sample $x(n-1)$ is stored in memory at time $n-1$ and it is recalled from memory at time n to form

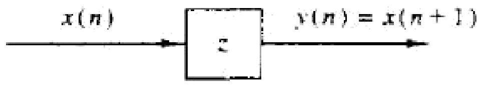
$$y(n) = x(n-1)$$

Thus this basic building block requires memory. The use of the symbol z^{-1} to denote the unit of delay will become apparent when we discuss the z-transform in Chapter 3.



Graphical representation of the unit delay element.

A unit advance element. In contrast to the unit delay, a unit advance moves the input $x(n)$ ahead by one sample in time to yield $x(n+1)$. Figure 2.17 illustrates this operation, with the operator z being used to denote the unit advance.



Graphical representation of the unit advance element.

We observe that any such advance is physically impossible in real time, since, in fact, it involves looking into the future of the signal. On the other hand, if we store the signal in the memory of the computer, we can recall any sample at any time. In such a nonreal-time application, it is possible to advance the signal $x(n)$ in time.

Example 2.2.3

Using basic building blocks introduced above, sketch the block diagram representation of the discrete-time system described by the input–output relation.

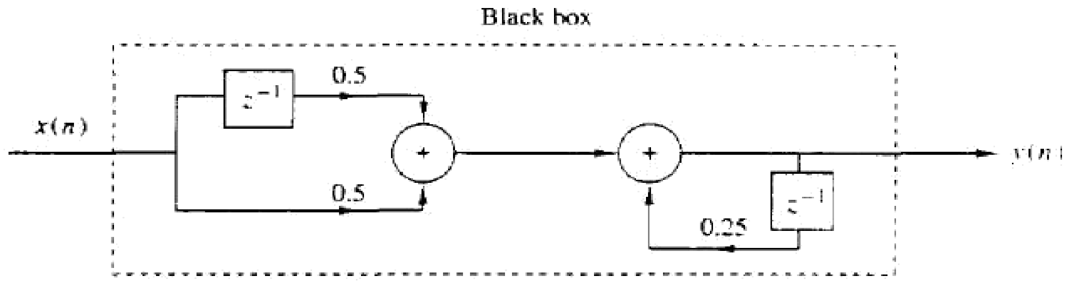
$$y(n) = \tfrac{1}{4}y(n-1) + \tfrac{1}{2}x(n) + \tfrac{1}{2}x(n-1)$$

where $x(n)$ is the input and $y(n)$ is the output of the system.

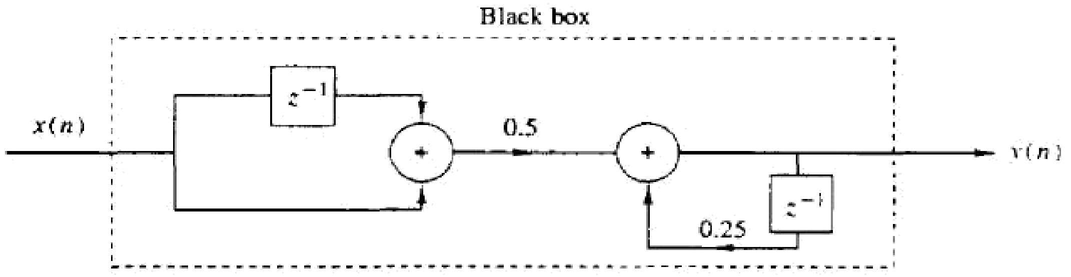
Solution According to (2.2.5), the output $y(n)$ is obtained by multiplying the input $x(n)$ by 0.5, multiplying the previous input $x(n-1)$ by 0.5, adding the two products, and then adding the previous output $y(n-1)$ multiplied by $\frac{1}{4}$. Figure 2.18a illustrates this block diagram realization of the system. A simple rearrangement of (2.2.5), namely,

$$y(n) = \tfrac{1}{4}y(n-1) + \tfrac{1}{2}[x(n) + x(n-1)]$$

leads to the block diagram realization shown in Fig. 2.18b. Note that if we treat "the system" from the "viewpoint" of an input–output or an external description, we are not concerned about how the system is realized. On the other hand, if we adopt an



(a)



(b)

Block diagram realizations of the system $y(n) = 0.25y(n-1) + 0.5x(n) + 0.5x(n-1)$.

internal description of the system, we know exactly how the system building blocks are configured. In terms of such a realization, we can see that a system is *relaxed* at time $n = n_0$ if the outputs of all the *delays* existing in the system are zero at $n = n_0$ (i.e., all memory is *filled* with zeros).

Classification of Discrete-Time Systems

In the analysis as well as in the design of systems, it is desirable to classify the systems according to the general properties that they satisfy. In fact, the mathematical techniques that we develop in this and in subsequent chapters for analyzing and designing discrete-time systems depend heavily on the general characteristics of the systems that are being considered. For this reason it is necessary for us to develop a number of properties or categories that can be used to describe the general characteristics of systems.

We stress the point that for a system to possess a given property, the property must hold for every possible input signal to the system. If a property holds for some input signals but not for others, the system does not possess that property. Thus a counterexample is sufficient to prove that a system does not possess a property. However, to prove that the system has some property, we must prove that this property holds for every possible input signal.

Static versus dynamic systems. A discrete-time system is called *static* or memoryless if its output at any instant n depends at most on the input sample at the same time, but not on past or future samples of the input. In any other case, the system is said to be *dynamic* or to have memory. If the output of a system at time n is completely determined by the input samples in the interval from $n - N$ to $n (N \geq 0)$, the system is said to have *memory* of duration N. If $N = 0$, the system is static. If $0 < N < \infty$, the system is said to have *finite memory*, whereas if $N = \infty$, the system is said to have *infinite memory*.

The systems described by the following input–output equations

$$y(n) = ax(n)$$

$$y(n) = nx(n) + bx^3(n)$$

are both static or memoryless. Note that there is no need to store any of the past inputs or outputs in order to compute the present output. On the other hand, the systems described by the following input–output relations

$$y(n) = x(n) + 3x(n-1)$$

$$y(n) = \sum_{k=0}^{n} x(n-k)$$

$$y(n) = \sum_{k=0}^{\infty} x(n-k)$$

are dynamic systems or systems with memory. The systems described by and (2.2.10) have finite memory, whereas the system described by (2.2.11) has infinite memory.

We observe that static or memoryless systems are described in general by input–output equations of the form

$$y(n) = \mathcal{T}[x(n), n]$$

and they do not include delay elements (memory).

Time-invariant versus time-variant systems. We can subdivide the general class of systems into the two broad categories, time-invariant systems and time-variant systems. A system is called time-invariant if its input–output characteristics do not change with time. To elaborate, suppose that we have a system $\mathcal{T}$ in a relaxed state which, when excited by an input signal $x(n)$, produces an output signal $y(n)$. Thus we write

$$y(n) = \mathcal{T}[x(n)]$$

Now suppose that the same input signal is delayed by k units of time to yield $x(n-k)$, and again applied to the same system. If the characteristics of the system do not change with time, the output of the relaxed system will be $y(n-k)$. That is, the output will be the same as the response to $x(n)$, except that it will be delayed by the same k units in time that the input was delayed. This leads us to define a time-invariant or shift-invariant system as follows.

Definition. A relaxed system $\mathcal{T}$ is *time invariant* or *shift invariant* if and only if

$$x(n) \xrightarrow{\mathcal{T}} y(n)$$

implies that

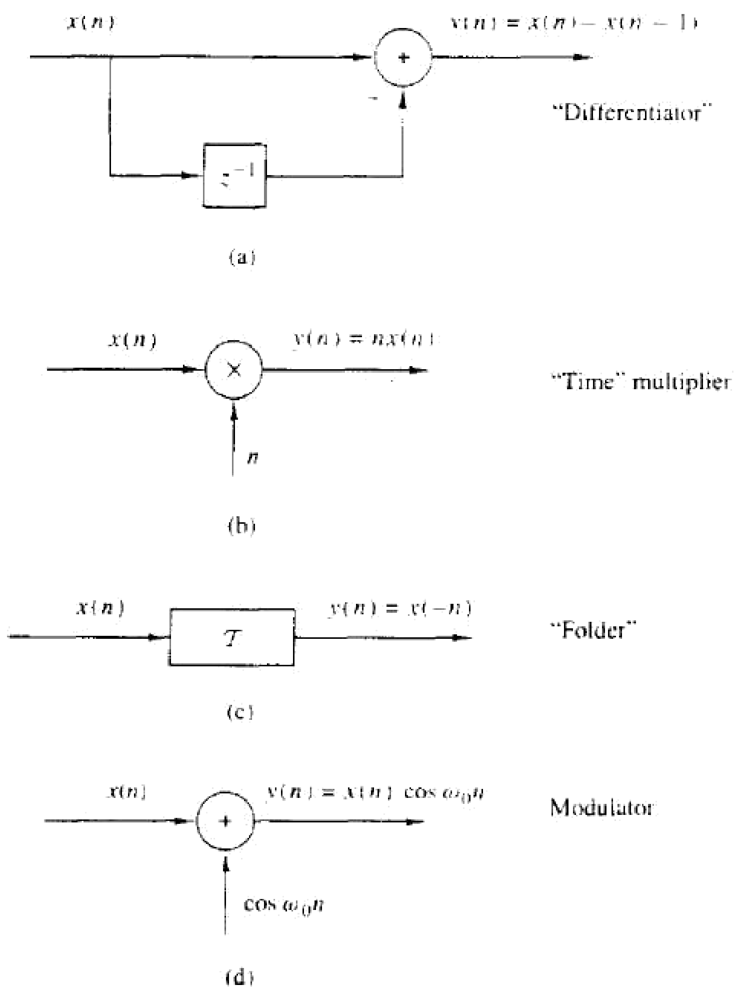
$$x(n-k) \xrightarrow{\mathcal{T}} y(n-k)$$

for every input signal $x(n)$ and every time shift k.

To determine if any given system is time invariant, we need to perform the test specified by the preceding definition. Basically, we excite the system with an arbitrary input sequence $x(n)$, which produces an output denoted as $y(n)$. Next we delay the input sequence by same amount k and recompute the output. In general, we can write the output as

$$y(n,k) = \mathcal{T}[x(n-k)]$$

Now if this output $y(n,k) = y(n-k)$, for all possible values of k, the system is time invariant. On the other hand, if the output $y(n,k) \neq y(n-k)$, even for one value of k, the system is time variant.



Examples of a time-invariant (a) and some time-variant systems (b)–(d).

Example 2.2.4

Determine if the systems shown in Fig. 2.19 are time invariant or time variant.

Solution

(a) This system is described by the input–output equations

$$y(n) = \mathcal{T}[x(n)] = x(n) - x(n-1)$$

Now if the input is delayed by k units in time and applied to the system, it is clear from the block diagram that the output will be

$$y(n, k) = x(n-k) - x(n-k-1)$$

On the other hand, from (2.2.14) we note that if we delay $y(n)$ by k units in time, we obtain

$$y(n-k) = x(n-k) - x(n-k-1)$$

Since the right-hand sides of (2.2.16) and (2.2.17) are identical, it follows that $y(n, k) = y(n-k)$. Therefore, the system is time invariant.

(b) The input–output equation for this system is

$$y(n) = \mathcal{T}[x(n)] = nx(n)$$

The response of this system to $x(n-k)$ is

$$y(n,k) = nx(n-k)$$

Now if we delay $y(n)$ in (2.2.18) by k units in time, we obtain

$$\begin{aligned} y(n-k) &= (n-k)x(n-k) \\ &= nx(n-k) - kx(n-k) \end{aligned}$$

This system is time variant, since $y(n,k) \neq y(n-k)$.

(c) This system is described by the input–output relation

$$y(n) = \mathcal{T}[x(n)] = x(-n)$$

The response of this system to $x(n-k)$ is

$$y(n,k) = \mathcal{T}[x(n-k)] = x(-n-k)$$

Now, if we delay the output $y(n)$, as given by (2.2.21), by k units in time, the result will be

$$y(n-k) = x(-n+k)$$

Since $y(n,k) \neq y(n-k)$, the system is time variant.

(d) The input–output equation for this system is

$$y(n) = x(n)\cos\omega_0 n$$

The response of this system to $x(n-k)$ is

$$y(n,k) = x(n-k)\cos\omega_0 n$$

If the expression in (2.2.24) is delayed by k units and the result is compared to (2.2.25), it is evident that the system is time variant.

Linear versus nonlinear systems. The general class of systems can also be subdivided into linear systems and nonlinear systems. A linear system is one that satisfies the *superposition principle.* Simply stated, the principle of superposition requires that the response of the system to a weighted sum of signals be equal to the corresponding weighted sum of the responses (outputs) of the system to each of the individual input signals. Hence we have the following definition of linearity.

Definition. A relaxed $\mathcal{T}$ system is linear if and only if

$$\mathcal{T}[a_1x_1(n) + a_2x_2(n)] = a_1\mathcal{T}[x_1(n)] + a_2\mathcal{T}[x_2(n)]$$

for any arbitrary input sequences $x_1(n)$ and $x_2(n)$, and any arbitrary constants a_1 and a_2.

Figure 2.20 gives a pictorial illustration of the superposition principle.

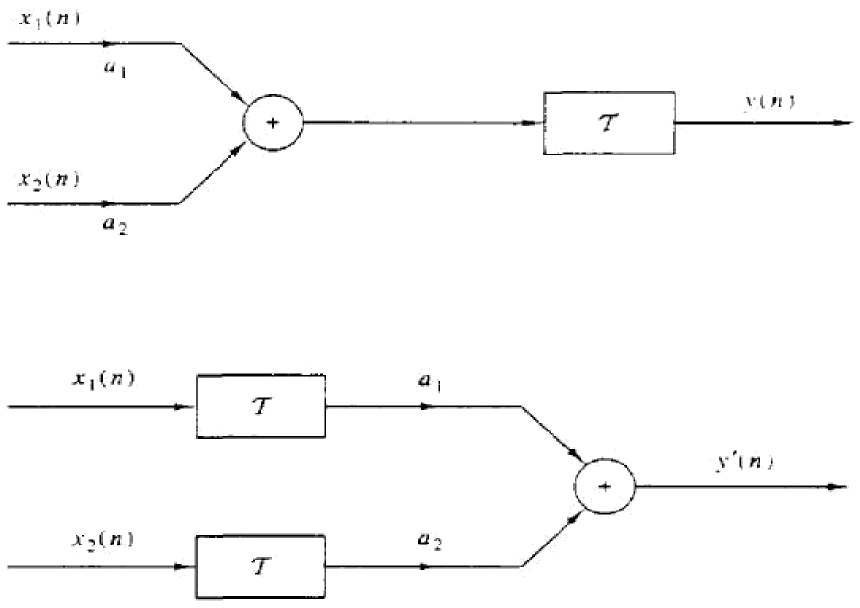


Figure 2.20 Graphical representation of the superposition principle. $\mathcal{T}$ is linear if and only if $y(n) = y'(n)$.

The superposition principle embodied in the relation (2.2.26) can be separated into two parts. First, suppose that $a_2 = 0$. Then (2.2.26) reduces to

$$\mathcal{T}[a_1 x_1(n)] = a_1 \mathcal{T}[x_1(n)] = a_1 y_1(n)$$

where

$$y_1(n) = \mathcal{T}[x_1(n)]$$

The relation (2.2.27) demonstrates the *multiplicative* or *scaling property* of a linear system. That is, if the response of the system to the input $x_1(n)$ is $y_1(n)$, the response to $a_1 x_1(n)$ is simply $a_1 y_1(n)$. Thus any scaling of the input results in an identical scaling of the corresponding output.

Second, suppose that $a_1 = a_2 = 1$ in (2.2.26). Then

$$\begin{aligned}\mathcal{T}[x_1(n) + x_2(n)] &= \mathcal{T}[x_1(n)] + \mathcal{T}[x_1(n)] \\ &= y_1(n) + y_2(n)\end{aligned}$$

This relation demonstrates the *additivity property* of a linear system. The additivity and multiplicative properties constitute the superposition principle as it applies to linear systems.

The linearity condition embodied in (2.2.26) can be extended arbitrarily to any weighted linear combination of signals by induction. In general, we have

$$x(n) = \sum_{k=1}^{M-1} a_k x_k(n) \xrightarrow{\mathcal{T}} y(n) = \sum_{k=1}^{M-1} a_k y_k(n)$$

where

$$y_k(n) = \mathcal{T}[x_k(n)] \qquad k = 1, 2, \ldots, M-1$$

We observe from (2.2.27) that if $a_1 = 0$, then $y(n) = 0$. In other words, a relaxed, linear system with zero input produces a zero output. If a system produces a nonzero output with a zero input, the system may be either nonrelaxed or nonlinear. If a relaxed system does not satisfy the superposition principle as given by the definition above, it is called *nonlinear*.

Example 2.2.5

Determine if the systems described by the following input–output equations are linear or nonlinear.

(a) $y(n) = nx(n)$ **(b)** $y(n) = x(n^2)$ **(c)** $y(n) = x^2(n)$

(d) $y(n) = Ax(n) + B$ **(e)** $y(n) = e^{x(n)}$

Solution

(a) For two input sequences $x_1(n)$ and $x_2(n)$, the corresponding outputs are

$$y_1(n) = nx_1(n)$$
$$y_2(n) = nx_2(n)$$

A linear combination of the two input sequences results in the output

$$\begin{aligned} y_3(n) &= T[a_1x_1(n) + a_2x_2(n)] = n[a_1x_1(n) + a_2x_2(n)] \\ &= a_1nx_1(n) + a_2nx_2(n) \end{aligned}$$

On the other hand, a linear combination of the two outputs in (2.2.31) results in the output

$$a_1y_1(n) + a_2y_2(n) = a_1nx_1(n) + a_2nx_2(n)$$

Since the right-hand sides of (2.2.32) and (2.2.33) are identical, the system is linear.

(b) As in part (a), we find the response of the system to two separate input signals $x_1(n)$ and $x_2(n)$. The result is

$$y_1(n) = x_1(n^2)$$
$$y_2(n) = x_2(n^2)$$

The output of the system to a linear combination of $x_1(n)$ and $x_2(n)$ is

$$y_3(n) = T[a_1x_1(n) + a_2x_2(n)] = a_1x_1(n^2) + a_2x_2(n^2)$$

Finally, a linear combination of the two outputs in (2.2.36) yields

$$a_1y_1(n) + a_2y_2(n) = a_1x_1(n^2) + a_2x_2(n^2)$$

By comparing (2.2.35) with (2.2.36), we conclude that the system is linear.

(c) The output of the system is the square of the input. (Electronic devices that have such an input–output characteristic and are called square-law devices.) From our previous discussion it is clear that such a system is memoryless. We now illustrate that this system is nonlinear.

The responses of the system to two separate input signals are

$$y_1(n) = x_1^2(n)$$

$$y_2(n) = x_2^2(n)$$

The response of the system to a linear combination of these two input signals is

$$\begin{aligned} y_3(n) &= \mathcal{T}[a_1x_1(n) + a_2x_2(n)] \\ &= [a_1x_1(n) + a_2x_2(n)]^2 \\ &= a_1^2x_1^2(n) + 2a_1a_2x_1(n)x_2(n) + a_2^2x_2^2(n) \end{aligned}$$

On the other hand, if the system is linear, it would produce a linear combination of the two outputs in (2.2.37), namely,

$$a_1y_1(n) + a_2y_2(n) = a_1x_1^2(n) + a_2x_2^2(n)$$

Since the actual output of the system, as given by (2.2.38), is not equal to (2.2.39), the system is nonlinear.

(d) Assuming that the system is excited by $x_1(n)$ and $x_2(n)$ separately, we obtain the corresponding outputs

$$y_1(n) = Ax_1(n) + B$$

$$y_2(n) = Ax_2(n) + B$$

A linear combination of $x_1(n)$ and $x_2(n)$ produces the output

$$\begin{aligned} y_3(n) &= \mathcal{T}[a_1x_1(n) + a_2x_2(n)] \\ &= A[a_1x_1(n) + a_2x_2(n)] + B \\ &= Aa_1x_1(n) + a_2Ax_2(n) + B \end{aligned}$$

On the other hand, if the system were linear, its output to the linear combination of $x_1(n)$ and $x_2(n)$ would be a linear combination of $y_1(n)$ and $y_2(n)$, that is,

$$a_1y_1(n) + a_2y_2(n) = a_1Ax_1(n) + a_1B + a_2Ax_2(n) + a_2B$$

Clearly, (2.2.41) and (2.2.42) are different and hence the system fails to sa[illegible]fy the linearity test.

The reason that this system fails to satisfy the linearity test is not that the system is nonlinear (in fact, the system is described by a linear equation) but the presence of the constant B. Consequently, the output depends on both the input excitation and on the parameter $B \neq 0$. Hence, for $B \neq 0$, the system is not relaxed. If we set $B = 0$, the system is now relaxed and the linearity test is satisfied.

(e) Note that the system described by the input–output equation

$$y(n) = e^{x(n)}$$

is relaxed. If $x(n) = 0$, we find that $y(n) = 1$. This is an indication that the system is nonlinear. This, in fact, is the conclusion reached when the linearity test, is applied.

Causal versus noncausal systems. We begin with the definition of causal iscrete-time systems.

and the output of the second system is

$$\begin{aligned} y(n) &= \mathcal{T}_2[y_1(n)] \\ &= \mathcal{T}_2\{\mathcal{T}_1[x(n)]\} \end{aligned}$$

We observe that systems $\mathcal{T}_1$ and $\mathcal{T}_2$ can be combined or consolidated into a single overall system

$$\mathcal{T}_c \equiv \mathcal{T}_2\mathcal{T}_1$$

Consequently, we can express the output of the combined system as

$$y(n) = \mathcal{T}_c[x(n)]$$

In general, the order in which the operations $\mathcal{T}_1$ and $\mathcal{T}_2$ are performed is important. That is,

$$\mathcal{T}_2\mathcal{T}_1 \neq \mathcal{T}_1\mathcal{T}_2$$

for arbitrary systems. However, if the systems $\mathcal{T}_1$ and $\mathcal{T}_2$ are linear and time invariant, then (a) $\mathcal{T}_c$ is time invariant and (b) $\mathcal{T}_2\mathcal{T}_1 = \mathcal{T}_1\mathcal{T}_2$, that is, the order in which the systems process the signal is not important. $\mathcal{T}_2\mathcal{T}_1$ and $\mathcal{T}_1\mathcal{T}_2$ yield identical output sequences.

The proof of (a) follows. The proof of (b) is given in Section 2.3.4. To prove time invariance, suppose that $\mathcal{T}_1$ and $\mathcal{T}_2$ are time invariant; then

$$x(n-k) \xrightarrow{\mathcal{T}_1} y_1(n-k)$$

and

$$y_1(n-k) \xrightarrow{\mathcal{T}_2} y(n-k)$$

Thus

$$x(n-k) \xrightarrow{\mathcal{T}_c=\mathcal{T}_2\mathcal{T}_1} y(n-k)$$

and therefore, $\mathcal{T}_c$ is time invariant.

In the parallel interconnection, the output of the system $\mathcal{T}_1$ is $y_1(n)$ and the output of the system $\mathcal{T}_2$ is $y_2(n)$. Hence the output of the parallel interconnection is

$$\begin{aligned} y_3(n) &= y_1(n) + y_2(n) \\ &= \mathcal{T}_1[x(n)] + \mathcal{T}_2[x(n)] \\ &= (\mathcal{T}_1 + \mathcal{T}_2)[x(n)] \\ &= \mathcal{T}_p[x(n)] \end{aligned}$$

where $\mathcal{T}_p = \mathcal{T}_1 + \mathcal{T}_2$.

In general, we can use parallel and cascade interconnection of systems to construct larger, more complex systems. Conversely, we can take a larger system and break it down into smaller subsystems for purposes of analysis and implementation. We shall use these notions later, in the design and implementation of digital filters.

Definition. A system is said to be *causal* if the output of the system at any time n [i.e., $y(n)$] depends only on present and past inputs [i.e., $x(n)$, $x(n-1)$, $x(n-2), \ldots$], but does not depend on future inputs [i.e., $x(n+1)$, $x(n+2), \ldots$]. In mathematical terms, the output of a causal system satisfies an equation of the form

$$y(n) = F[x(n), x(n-1), x(n-2), \ldots] \tag{2.2.44}$$

where $F[\cdot]$ is some arbitrary function.

If a system does not satisfy this definition, it is called *noncausal*. Such a system has an output that depends not only on present and past inputs but also on future inputs.

It is apparent that in real-time signal processing applications we cannot observe future values of the signal, and hence a noncausal system is physically unrealizable (i.e., it cannot be implemented). On the other hand, if the signal is recorded so that the processing is done off-line (nonreal time), it is possible to implement a noncausal system, since all values of the signal are available at the time of processing. This is often the case in the processing of geophysical signals and images.

Example 2.2.6

Determine if the systems described by the following input–output equations are causal or noncausal.

(a) $y(n) = x(n) - x(n-1)$ **(b)** $y(n) = \sum_{k=-\infty}^{n} x(k)$ **(c)** $y(n) = ax(n)$

(d) $y(n) = x(n) + 3x(n+4)$ **(e)** $y(n) = x(n^2)$ **(f)** $y(n) = x(2n)$

(g) $y(n) = x(-n)$

Solution The systems described in parts (a), (b), and (c) are clearly causal, since the output depends only on the present and past inputs. On the other hand, the systems in parts (d), (e), and (f) are clearly noncausal, since the output depends on future values of the input. The system in (g) is also noncausal, as we note by selecting, for example, $n = -1$, which yields $y(-1) = x(1)$. Thus the output at $n = -1$ depends on the input at $n = 1$, which is two units of time into the future.

Stable versus unstable systems. Stability is an important property that must be considered in any practical application of a system. Unstable systems usually exhibit erratic and extreme behavior and cause overflow in any practical implementation. Here, we define mathematically what we mean by a stable system, and later, in Section 2.3.6, we explore the implications of this definition for linear, time-invariant systems.

Definition. An arbitrary relaxed system is said to be bounded input–bounded output (BIBO) stable if and only if every bounded input produces a bounded output.

The conditions that the input sequence $x(n)$ and the output sequence $y(n)$ are bounded is translated mathematically to mean that there exist some finite numbers, say M_x and M_y, such that

$$|x(n)| \le M_x < \infty \qquad |y(n)| \le M_y < \infty \tag{2.2.45}$$

for all n. If, for some bounded input sequence $x(n)$, the output is unbounded (infinite), the system is classified as unstable.

Example 2.2.7

Consider the nonlinear system described by the input–output equation

$$y(n) = y^2(n-1) + x(n)$$

As an input sequence we select the bounded signal

$$x(n) = C\delta(n)$$

where C is a constant. We also assume that $y(-1) = 0$. Then the output sequence is

$$y(0) = C, \quad y(1) = C^2, \quad y(2) = C^4, \quad \ldots, \quad y(n) = C^{2^n}$$

Clearly, the output is unbounded when $1 < |C| < \infty$. Therefore, the system is BIBO unstable, since a bounded input sequence has resulted in an unbounded output.

Interconnection of Discrete-Time Systems

Discrete-time systems can be interconnected to form larger systems. There are two basic ways in which systems can be interconnected: in cascade (series) or in parallel. These interconnections are illustrated in Fig. 2.21. Note that the two interconnected systems are different.

In the cascade interconnection the output of the first system is

$$y_1(n) = \mathcal{T}_1[x(n)]$$

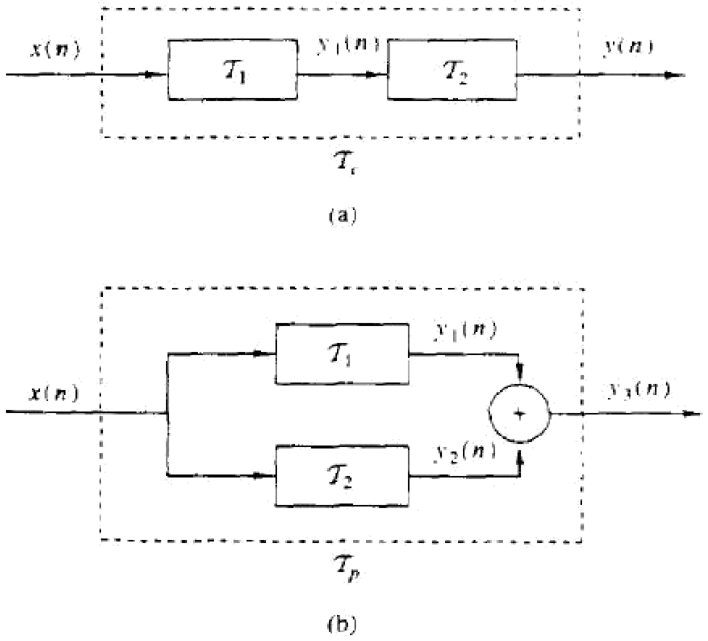
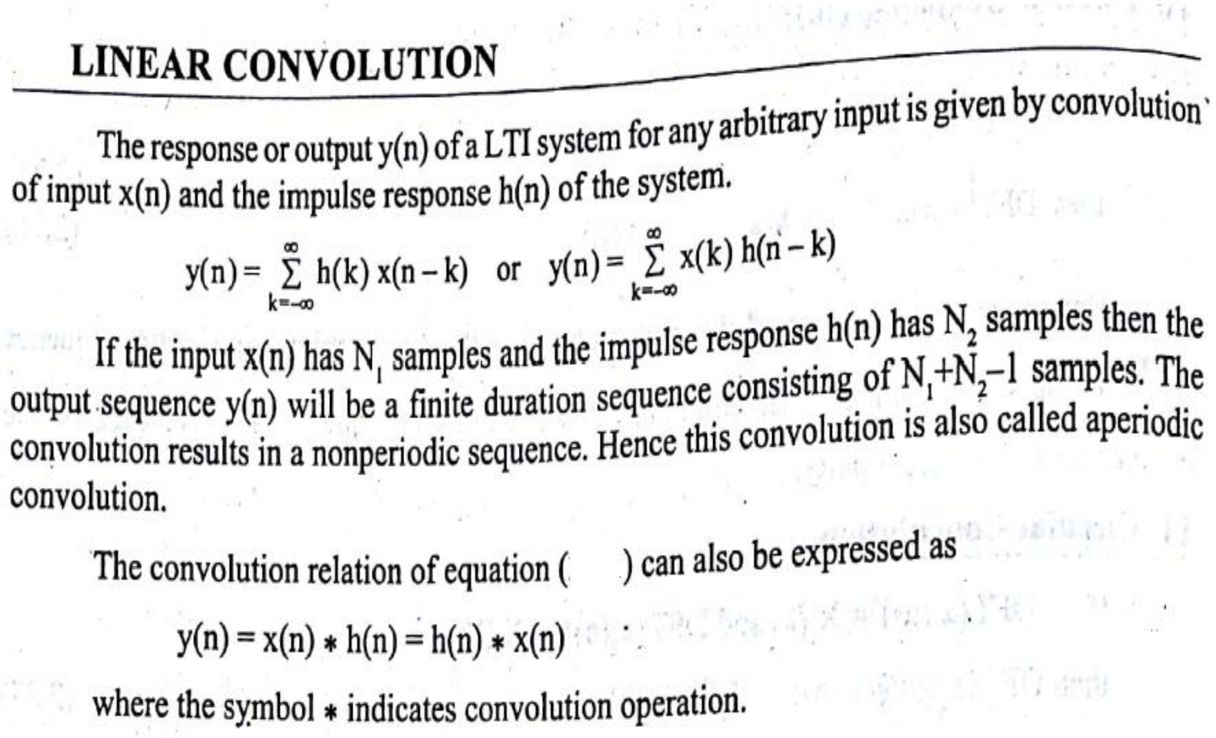


Figure 2.21 Cascade (a) and parallel (b) interconnections of systems.

Convolution

Convolution, one of the most important concepts in electrical engineering, can be used to determine the output a system produces for a given input signal. It can be shown that a linear time invariant system is completely characterized by its impulse response. The sifting property of the discrete time impulse function tells us that the input signal to a system can be represented as a sum of scaled and shifted unit impulses. Thus, by linearity, it would seem reasonable to compute of the output signal as the sum of scaled and shifted unit impulse responses. That is exactly what the operation of convolution accomplishes. Hence, convolution can be used to determine a linear time invariant system's output from knowledge of the input and the impulse response



LINEAR CONVOLUTION

The response or output y(n) of a LTI system for any arbitrary input is given by convolution of input x(n) and the impulse response h(n) of the system.

$$y(n) = \sum_{k=-\infty}^{\infty} h(k)\, x(n-k) \quad \text{or} \quad y(n) = \sum_{k=-\infty}^{\infty} x(k)\, h(n-k)$$

If the input x(n) has N_1 samples and the impulse response h(n) has N_2 samples then the output sequence y(n) will be a finite duration sequence consisting of N_1+N_2-1 samples. The convolution results in a nonperiodic sequence. Hence this convolution is also called aperiodic convolution.

The convolution relation of equation (   ) can also be expressed as

$$y(n) = x(n) * h(n) = h(n) * x(n)$$

where the symbol $*$ indicates convolution operation.

PROCEDURE FOR EVALUATING LINEAR CONVOLUTION

When we want to compute the output of the system at some time instant, say $n = n_0$, then according to convolution sum formula, the value of output sample at $n = n_0$ is given by

$$y(n_0) = \sum_{k=-\infty}^{+\infty} x(k)\, h(n_0 - k)$$

where $x(k)$ is the input sequence, and $h(k)$ is the impulse response of the system.

The index in the summation is k. Hence $x(n)$ is taken as $x(k)$ and $h(n)$ is taken as $h(k)$. [i.e., both the input signal $x(k)$ and the impulse response $h(n_0 - k)$ are functions of k].

The process of computing the convolution between $x(k)$ and $h(k)$ involves the following four steps.

1. Folding	:	Fold $h(k)$ about $k = 0$, to obtain $h(-k)$
2. Shifting	:	Shift $h(-k)$ by n_0 to the right if n_0 is positive, shift $h(-k)$ by n_0 to the left if n_0 is negative to obtain $h(n_0-k)$.
3. Multiplication	:	Multiply $x(k)$ by $h(n_0-k)$ to obtain the product sequence $v_{n0}(k) = x(k) \times h(n_0-k)$.
4. Summation	:	Sum all the values of the product sequence $v_{n0}(k)$ to obtain the value of the output at time $n = n_0$. [i.e., $y(n_0)$].

The above procedure results in the response of the system at a single time instant say $n = n_0$. In general we are interested in evaluating the response of the system over all the time instants $-\infty < n < \infty$. Hence the steps 2 through 4 given above must be repeated, for all possible time shifts in the range $-\infty < n < \infty$.

If $x(n)$ starts at $n = n_x$ and $h(n)$ starts at $n = n_h$ then the initial value of n for $y(n)$ is $\mathbf{n = n_x + n_h}$. The value of $x(n)$ for $n < n_x$ and the value of $h(n)$ for $n < n_h$ are then assumed to be zero.

If length of $x(n)$ is N_1 and length of $h(n)$ is N_2 then the length of $y(n)$ is N_1+N_2-1 and the final value of n for $y(n)$ is $\mathbf{n = (n_x+n_h) + (N_1+N_2-2)}$.

PROPERTIES OF LINEAR CONVOLUTION

1. Commutative Property

$$y(n) = x(n) * h(n) = h(n) * x(n)$$

Proof

We know that, $y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)\, h(n-k)$

Let $m = n-k$, then $k = n-m$. On substituting k by n–m and m by n–k in the above equation, we have

$$y(n) = \sum_{m=-\infty}^{\infty} x(n-m)\, h(m)$$

Since m is dummy index, we can replace m by k

$$y(n) = \sum_{k=-\infty}^{\infty} x(n-k)\, h(k) = \sum_{k=-\infty}^{\infty} h(k)\, x(n-k) = h(n) * x(n)$$

2. Associative Property

$$[x(n) * h_1(n)] * h_2(n) = x(n) * [h_1(n) * h_2(n)]$$

Proof

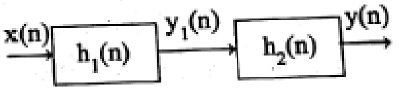
To prove left hand side (LHS) of the property

Let x(n) be the input signal to a LTI system with impulse response $h_1(n)$.

Now $y_1(n) = x(n) * h_1(n)$.

The signal $y_1(n)$ is now input to a second LTI system with impulse response $h_2(n)$. Then the output of second system is given by

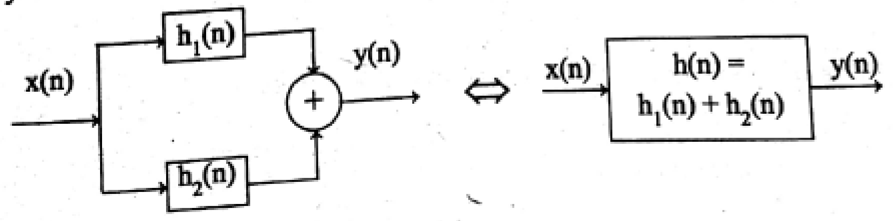
$$y(n) = y_1(n) * h_2(n) = [x(n) * h_1(n)] * h_2(n)$$



3. Distributive Property

$$x(n) * [h_1(n) + h_2(n)] = [x(n) * h_1(n)] + [x(n) * h_2(n)]$$

This law implies that if two LTI systems with impulse response $h_1(n)$ and $h_2(n)$ are excited by the same input signal x(n), the sum of the two responses is identical to the response of an overall system with impulse response, $h(n) = h_1(n) + h_2(n)$.



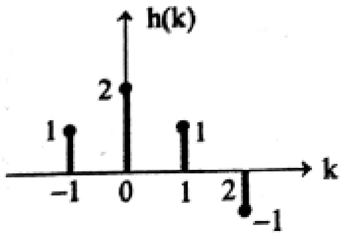
Determine the response of the system whose input x(n) and impulse response h(n) are given by $x(n) = \{\underset{\uparrow}{1}, 2, 3, 1\}$ and $h(n) = \{1, \underset{\uparrow}{2}, 1, -1\}$

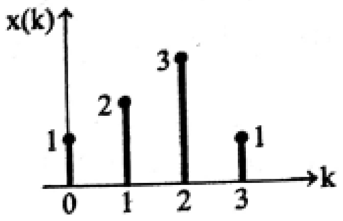
SOLUTION

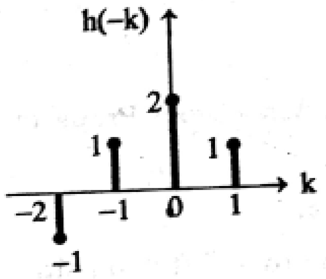
The response y(n) of the system is given by convolution of x(n) and h(n)

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{+\infty} x(k)\, h(n-k)$$

The graphical representation of x(n) and h(n) after replacing n by k are shown below.







The sequence h(k) is folded with respect to k = 0 to obtain h(–k). The Input sequence start at n = 0 and the impulse response sequence start at n = –1. Therefore the output sequence start at n = 0 + (–1) = –1. The input and impulse response consist of 4 samples, so the output consists of 4 + 4 – 1 = 7 samples. The 7 samples of y(n) are computed as shown below.

By convolution formula,

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)\, h(n-k) = \sum_{k=-\infty}^{+\infty} x(k)\, h_n(k)\,; \text{ where } h_n(k) = h(n-k)$$

$$\therefore \text{When } n = -1,\ y(-1) = \sum_{k=-\infty}^{\infty} x(k)\, h(-1-k) = \sum_{k=-\infty}^{\infty} x(k)\, h_{-1}(k)$$

$$\text{When } n = 0,\quad y(0) = \sum_{k=-\infty}^{\infty} x(k)\, h(0-k) = \sum_{k=-\infty}^{\infty} x(k)\, h_0(k)$$

$$\text{When } n = 1,\quad y(1) = \sum_{k=-\infty}^{\infty} x(k)\, h(1-k) = \sum_{k=-\infty}^{\infty} x(k)\, h_1(k)$$

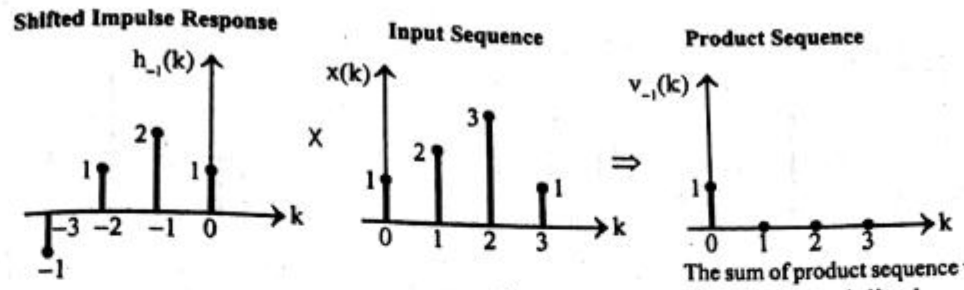
$$\text{When } n = 2,\quad y(2) = \sum_{k=-\infty}^{\infty} x(k)\, h(2-k) = \sum_{k=-\infty}^{\infty} x(k)\, h_2(k)$$

$$\text{When } n = 3,\quad y(3) = \sum_{k=-\infty}^{\infty} x(k)\, h(3-k) = \sum_{k=-\infty}^{\infty} x(k)\, h_3(k)$$

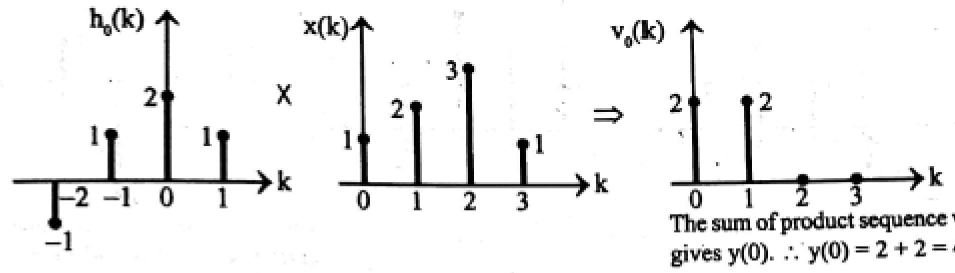
$$\text{When } n = 4,\quad y(4) = \sum_{k=-\infty}^{\infty} x(k)\, h(4-k) = \sum_{k=-\infty}^{\infty} x(k)\, h_4(k)$$

$$\text{When } n = 5,\quad y(5) = \sum_{k=-\infty}^{\infty} x(k)\, h(5-k) = \sum_{k=-\infty}^{\infty} x(k)\, h_5(k)$$

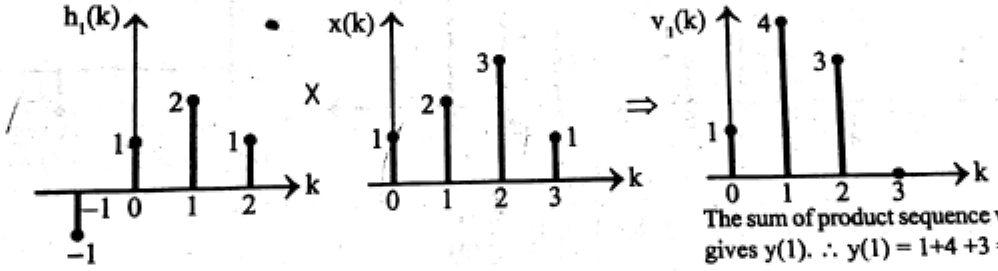
The computation of each sample is graphically shown below.



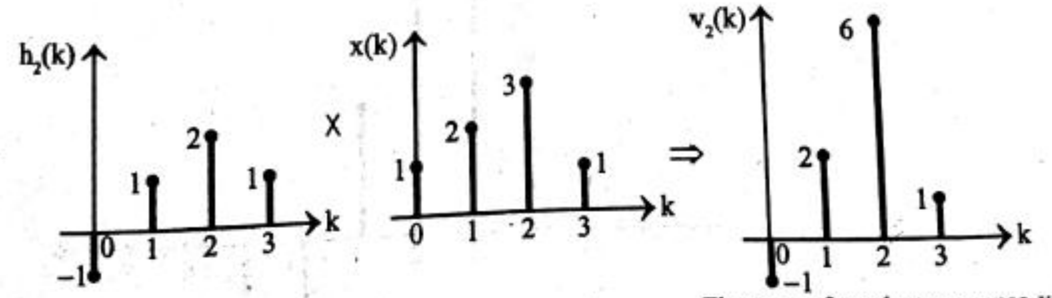
The sum of product sequence $v_{-1}(k)$ gives $y(-1)$. $\therefore$ $y(-1) = 1$.



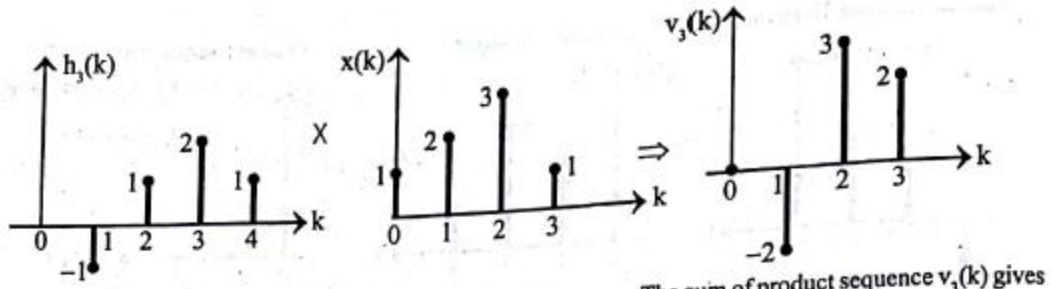
The sum of product sequence $v_0(k)$ gives $y(0)$. $\therefore$ $y(0) = 2 + 2 = 4$.

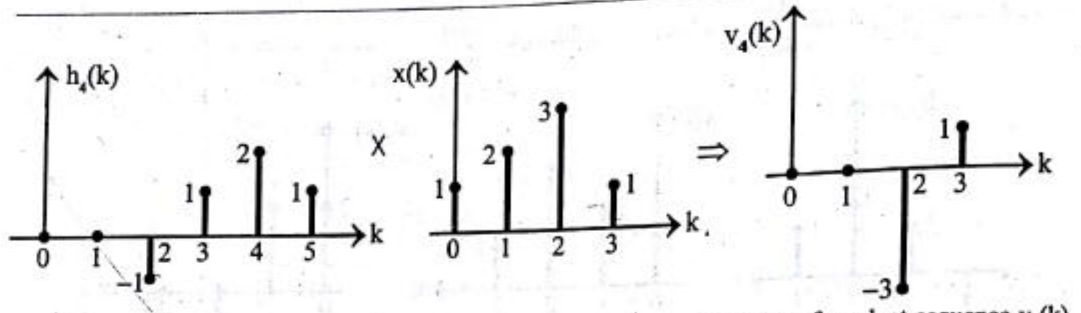


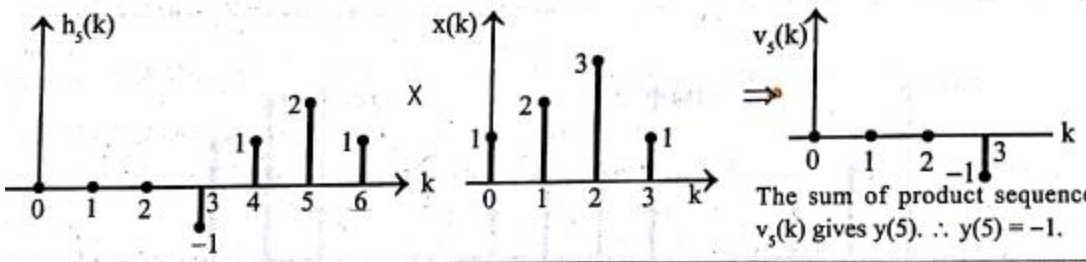
The sum of product sequence $v_1(k)$ gives $y(1)$. $\therefore$ $y(1) = 1+4+3 = 8$.



The sum of product sequence $v_2(k)$ gives $y(2)$. $\therefore$ $y(2) = -1+2+6+1 = 8$.


The sum of product sequence $v_3(k)$ gives $y(3)$. $\therefore$ $y(3) = -2+3+2 = 3$.


The sum of product sequence $v_4(k)$ gives $y(4)$. $\therefore$ $y(4) = -3+1 = -2$.


The sum of product sequence $v_5(k)$ gives $y(5)$. $\therefore$ $y(5) = -1$.

The output sequence, $y(n) = \{1, \underset{\uparrow}{4}, 8, 8, 3, -2, -1\}$

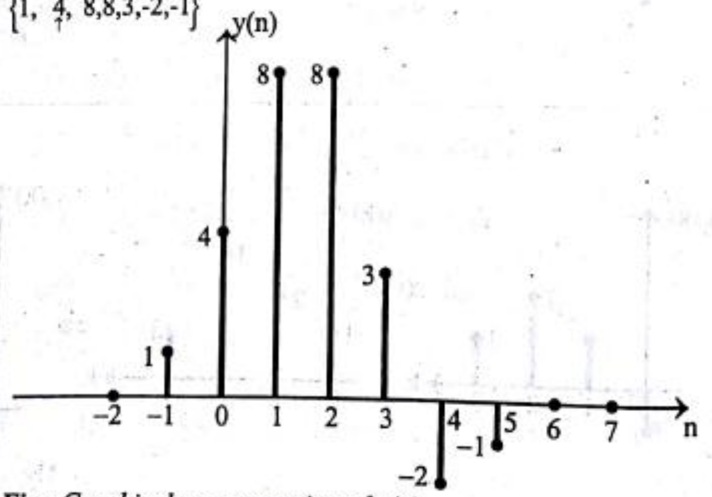


Fig : Graphical representation of y(n)

Analysis of DT-LTI Systems

The Direct z-Transform

The z-transform of a discrete-time signal $x(n)$ is defined as the power series

$$X(z) \equiv \sum_{n=-\infty}^{\infty} x(n) z^{-n}$$

where z is a complex variable. The relation (3.1.1) is sometimes called the *direct z-transform* because it transforms the time-domain signal $x(n)$ into its complex-plane representation $X(z)$. The inverse procedure [i.e., obtaining $x(n)$ from $X(z)$] is called the *inverse z-transform* and is examined briefly in Section 3.1.2 and in more detail in Section 3.4.

For convenience, the z-transform of a signal $x(n)$ is denoted by

$$X(z) \equiv Z\{x(n)\}$$

whereas the relationship between $x(n)$ and $X(z)$ is indicated by

$$x(n) \overset{z}{\longleftrightarrow} X(z)$$

Since the z-transform is an infinite power series, it exists only for those values of z for which this series converges. The *region of convergence* (ROC) of $X(z)$ is the set of all values of z for which $X(z)$ attains a finite value. Thus any time we cite a z-transform we should also indicate its ROC.

We illustrate these concepts by some simple examples.

Example 3.1.1

Determine the z-transforms of the following *finite-duration* signals.

(a) $x_1(n) = \{1, 2, 5, 7, 0, 1\}$
(b) $x_2(n) = \{1, 2, \underset{\uparrow}{5}, 7, 0, 1\}$
(c) $x_3(n) = \{0, 0, 1, 2, 5, 7, 0, 1\}$
(d) $x_4(n) = \{2, 4, \underset{\uparrow}{5}, 7, 0, 1\}$
(e) $x_5(n) = \delta(n)$
(f) $x_6(n) = \delta(n-k), k > 0$
(g) $x_7(n) = \delta(n+k), k > 0$

Solution From definition (3.1.1), we have

(a) $X_1(z) = 1 + 2z^{-1} + 5z^{-2} + 7z^{-3} + z^{-5}$, ROC: entire z-plane except $z = 0$
(b) $X_2(z) = z^2 + 2z + 5 + 7z^{-1} + z^{-3}$, ROC: entire z-plane except $z = 0$ and $z = \infty$
(c) $X_3(z) = z^{-2} + 2z^{-3} + 5z^{-4} + 7z^{-5} + z^{-7}$, ROC: entire z-plane except $z = 0$
(d) $X_4(z) = 2z^2 + 4z + 5 + 7z^{-1} + z^{-3}$, ROC: entire z-plane except $z = 0$ and $z = \infty$
(e) $X_5(z) = 1$[i.e., $\delta(n) \overset{z}{\longleftrightarrow} 1$], ROC: entire z-plane
(f) $X_6(z) = z^{-k}$[i.e., $\delta(n-k) \overset{z}{\longleftrightarrow} z^{-k}$], $k > 0$, ROC: entire z-plane except $z = 0$
(g) $X_7(z) = z^k$[*i.e.*, $\delta(n+k) \overset{z}{\longleftrightarrow} z^k$], $k > 0$, ROC: entire z-plane except $z = \infty$

.1 THE Z-TRANSFORM

In this section we introduce the z-transform of a discrete-time signal, investigate its convergence properties, and briefly discuss the inverse z-transform.

$$= \sum_{n=0}^{N/4-1} x(n) W_N^{kn} + W_N^{Nk/4} \sum_{n=0}^{N/4-1} x\left(n+\frac{N}{4}\right) W_N^{kn}$$
$$+ W_N^{kN/2} \sum_{n=0}^{N/4-1} x\left(n+\frac{N}{2}\right) W_N^{nk} + W_N^{3kN/4} \sum_{n=0}^{N/4-1} x\left(n+\frac{3N}{4}\right) W_N^{kn}$$

From the definition of the twiddle factors, we have

$$W_N^{kN/4} = (-j)^k \qquad W_N^{Nk/2} = (-1)^k \qquad W_N^{3Nk/4} = (j)^k$$

After substitution of (6.1.49) into (6.1.48), we obtain

$$X(k) = \sum_{n=0}^{N/4-1} \left[x(n) + (-j)^k x\left(n+\frac{N}{4}\right) \right.$$
$$\left. + (-1)^k x\left(n+\frac{N}{4}\right) + (j)^k x\left(n+\frac{3N}{4}\right) \right] W_N^{nk}$$

The relation in (6.1.50) is not an $N/4$-point DFT because the twiddle factor depends on N and not on $N/4$. To convert it into an $N/4$-point DFT, we subdivide the DFT sequence into four $N/4$-point subsequences, $X(4k)$, $X(4k+1)$, $X(4k+2)$, and $X(4k+3)$, $k = 0, 1, \ldots, N/4 - 1$. Thus we obtain the radix-4 decimation-in-frequency DFT as

$$X(4k) = \sum_{n=0}^{N/4-1} \left[x(n) + x\left(n+\frac{N}{4}\right) \right.$$
$$\left. + x\left(n+\frac{N}{2}\right) + x\left(n+\frac{3N}{4}\right) \right] W_N^0 W_{N/4}^{kn}$$

$$X(4k+1) = \sum_{n=0}^{N/4-1} \left[x(n) - jx\left(n+\frac{N}{4}\right) \right.$$
$$\left. - x\left(n+\frac{N}{2}\right) + jx\left(n+\frac{3N}{4}\right) \right] W_N^n W_{N/4}^{kn}$$

$$X(4k+2) = \sum_{n=0}^{N/4-1} \left[x(n) - x\left(n+\frac{N}{4}\right) \right.$$
$$\left. + x\left(n+\frac{N}{2}\right) - x\left(n+\frac{3N}{4}\right) \right] W_N^{2n} W_{N/4}^{kn}$$

$$X(4k+3) = \sum_{n=0}^{N/4-1} \left[x(n) + jx\left(n+\frac{N}{4}\right) \right.$$
$$\left. - x\left(n+\frac{N}{2}\right) - jx\left(n+\frac{3N}{4}\right) \right] W_N^{3n} W_{N/4}^{kn}$$

From this example it is easily seen that the ROC of a *finite-duration signal* is the entire z-plane, except possibly the points $z = 0$ and/or $z = \infty$. These points are excluded, because $z^k (k > 0)$ becomes unbounded for $z = \infty$ and $z^{-k} (k > 0)$ becomes unbounded for $z = 0$.

From a mathematical point of view the z-transform is simply an alternative representation of a signal. This is nicely illustrated in Example 3.1.1, where we see that the coefficient of z^{-n}, in a given transform, is the value of the signal at time n. In other words, the exponent of z contains the time information we need to identify the samples of the signal.

In many cases we can express the sum of the finite or infinite series for the z-transform in a closed-form expression. In such cases the z-transform offers a compact alternative representation of the signal.

Example 3.1.2

Determine the z-transform of the signal

$$x(n) = (\tfrac{1}{2})^n u(n)$$

Solution The signal $x(n)$ consists of an infinite number of nonzero values

$$x(n) = \{1, (\tfrac{1}{2}), (\tfrac{1}{2})^2, (\tfrac{1}{2})^3, \ldots, (\tfrac{1}{2})^n, \ldots\}$$

The z-transform of $x(n)$ is the infinite power series

$$\begin{aligned} X(z) &= 1 + \tfrac{1}{2}z^{-1} + (\tfrac{1}{2})^2 z^{-2} + (\tfrac{1}{2})^n z^{-n} + \cdots \\ &= \sum_{n=0}^{\infty} (\tfrac{1}{2})^n z^{-n} = \sum_{n=0}^{\infty} (\tfrac{1}{2}z^{-1})^n \end{aligned}$$

This is an infinite geometric series. We recall that

$$1 + A + A^2 + A^3 + \cdots = \frac{1}{1-A} \qquad \text{if } |A| < 1$$

Consequently, for $|\tfrac{1}{2}z^{-1}| < 1$, or equivalently, for $|z| > \tfrac{1}{2}$, $X(z)$ converges to

$$X(z) = \frac{1}{1 - \tfrac{1}{2}z^{-1}} \qquad \text{ROC: } |z| > \tfrac{1}{2}$$

We see that in this case, the z-transform provides a compact alternative representation of the signal $x(n)$.

Let us express the complex variable z in polar form as

$$z = re^{j\theta}$$

where $r = |z|$ and $\theta = \measuredangle z$. Then $X(z)$ can be expressed as

$$X(z)|_{z=re^{j\theta}} = \sum_{n=-\infty}^{\infty} x(n) r^{-n} e^{-j\theta n}$$

In the ROC of $X(z)$, $|X(z)| < \infty$. But

$$\begin{aligned}|X(z)| &= \left|\sum_{n=-\infty}^{\infty} x(n)r^{-n}e^{-j\theta n}\right| \\ &\le \sum_{n=-\infty}^{\infty} |x(n)r^{-n}e^{-j\theta n}| = \sum_{n=-\infty}^{\infty} |x(n)r^{-n}|\end{aligned}$$

Hence $|X(z)|$ is finite if the sequence $x(n)r^{-n}$ is absolutely summable.

The problem of finding the ROC for $X(z)$ is equivalent to determining the range of values of r for which the sequence $x(n)r^{-n}$ is absolutely summable. To elaborate, let us express (3.1.5) as

$$\begin{aligned}|X(z)| &\le \sum_{n=-\infty}^{-1} |x(n)r^{-n}| + \sum_{n=0}^{\infty} \left|\frac{x(n)}{r^n}\right| \\ &\le \sum_{n=1}^{\infty} |x(-n)r^{n}| + \sum_{n=0}^{\infty} \left|\frac{x(n)}{r^n}\right|\end{aligned}$$

If $X(z)$ converges in some region of the complex plane, both summations in (3.1.6) must be finite in that region. If the first sum in (3.1.6) converges, there must exist values of r small enough such that the product sequence $x(-n)r^n$, $1 \le n < \infty$, is absolutely summable. Therefore, the ROC for the first sum consists of all points in a circle of some radius r_1, where $r_1 < \infty$, as illustrated in Fig. 3.1a. On the other hand, if the second sum in (3.1.6) converges, there must exist values of r large enough such that the product sequence $x(n)/r^n$, $0 \le n < \infty$, is absolutely summable. Hence the ROC for the second sum in (3.1.6) consists of all points outside a circle of radius $r > r_2$, as illustrated in Fig. 3.1b.

Since the convergence of $X(z)$ requires that both sums in (3.1.6) be finite, it follows that the ROC of $X(z)$ is generally specified as the annular region in the z-plane, $r_2 < r < r_1$, which is the common region where both sums are finite. This region is illustrated in Fig. 3.1c. On the other hand, if $r_2 > r_1$, there is no common region of convergence for the two sums and hence $X(z)$ does not exist.

The following examples illustrate these important concepts.

Example 3.1.3

Determine the z-transform of the signal

$$x(n) = \alpha^n u(n) = \begin{cases}\alpha^n, & n \ge 0 \\ 0, & n < 0\end{cases}$$

Solution From the definition (3.1.1) we have

$$X(z) = \sum_{n=0}^{\infty} \alpha^n z^{-n} = \sum_{n=0}^{\infty} (\alpha z^{-1})^n$$

If $|\alpha z^{-1}| < 1$ or equivalently, $|z| > |\alpha|$, this power series converges to $1/(1 - \alpha z^{-1})$.

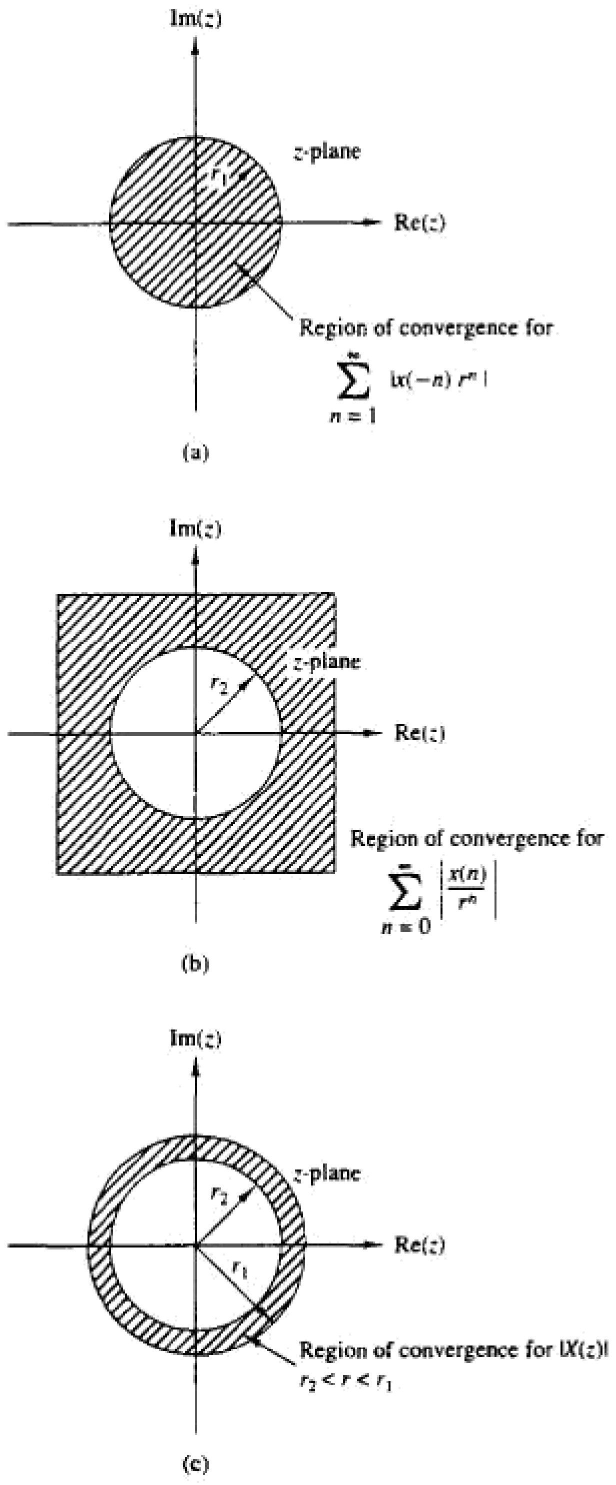

Im(z)
z-plane
r_1
Re(z)
Region of convergence for $\sum_{n=1}^{\infty} |x(-n)\, r^n|$
(a)
Im(z)
z-plane
r_2
Re(z)
Region of convergence for $\sum_{n=0}^{\infty} \left|\frac{x(n)}{r^n}\right|$
(b)
Im(z)
z-plane
r_2
r_1
Re(z)
Region of convergence for $|X(z)|$
$r_2 < r < r_1$
(c)

Figure 3.1 Region of convergence for $X(z)$ and its corresponding causal and anticausal components.

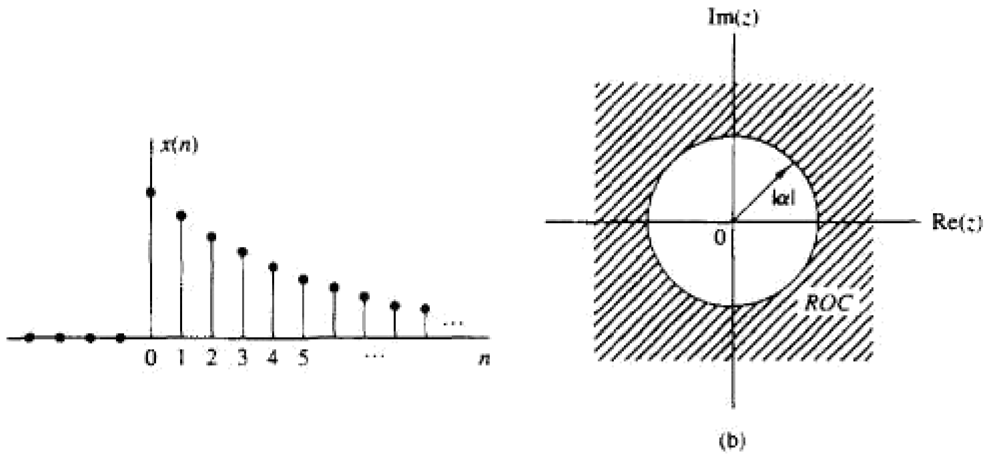


Figure 3.2 The exponential signal $x(n) = \alpha^n u(n)$ (a), and the ROC of its z-transform (b).

Thus we have the z-transform pair

$$x(n) = \alpha^n u(n) \overset{z}{\longleftrightarrow} X(z) = \frac{1}{1-\alpha z^{-1}} \qquad \text{ROC: } |z| > |\alpha|$$

The ROC is the exterior of a circle having radius $|\alpha|$. Figure 3.2 shows a graph of the signal $x(n)$ and its corresponding ROC. Note that, in general, α need not be real.

If we set $\alpha = 1$ in (3.1.7), we obtain the z-transform of the unit step signal

$$x(n) = u(n) \overset{z}{\longleftrightarrow} X(z) = \frac{1}{1-z^{-1}} \qquad \text{ROC: } |z| > 1$$

Example 3.1.4

Determine the z-transform of the signal

$$x(n) = -\alpha^n u(-n-1) = \begin{cases} 0, & n \geq 0 \\ -\alpha^n, & n \leq -1 \end{cases}$$

Solution From the definition (3.1.1) we have

$$X(z) = \sum_{n=-\infty}^{-1} (-\alpha^n) z^{-n} = -\sum_{l=1}^{\infty} (\alpha^{-1} z)^l$$

where $l = -n$. Using the formula

$$A + A^2 + A^3 + \cdots = A(1 + A + A^2 + \cdots) = \frac{A}{1-A}$$

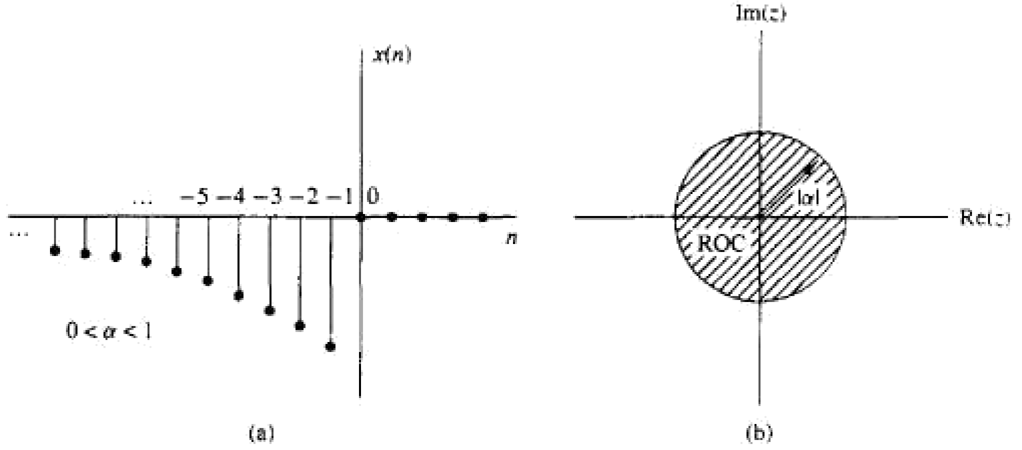
when $|A| < 1$ gives

$$X(z) = -\frac{\alpha^{-1} z}{1-\alpha^{-1} z} = \frac{1}{1-\alpha z^{-1}}$$

provided that $|\alpha^{-1} z| < 1$ or, equivalently, $|z| < |\alpha|$. Thus

$$x(n) = -\alpha^n u(-n-1) \overset{z}{\longleftrightarrow} X(z) = -\frac{1}{1-\alpha z^{-1}} \qquad \text{ROC: } |z| < |\alpha|$$

The ROC is now the interior of a circle having radius $|\alpha|$. This is shown in Fig. 3.3.



Anticausal signal $x(n) = -\alpha^n u(-n-1)$ (a), and the ROC of its z-transform (b).

Examples 3.1.3 and 3.1.4 illustrate two very important issues. The first concerns the uniqueness of the z-transform. From (3.1.7) and (3.1.9) we see that the causal signal $\alpha^n u(n)$ and the anticausal signal $-\alpha^n u(-n-1)$ have identical closed-form expressions for the z-transform, that is,

$$Z\{\alpha^n u(n)\} = Z\{-\alpha^n u(-n-1)\} = \frac{1}{1-\alpha z^{-1}}$$

This implies that a closed-form expression for the z-transform does not uniquely specify the signal in the time domain. The ambiguity can be resolved only if in addition to the closed-form expression, the ROC is specified. In summary, *a discrete-time signal $x(n)$ is uniquely determined by its z-transform $X(z)$ and the region of convergence of $X(z)$*. In this text the term "z-transform" is used to refer to both the closed-form expression and the corresponding ROC. Example 3.1.3 also illustrates the point that *the ROC of a causal signal is the exterior of a circle of some radius r_2 while the ROC of an anticausal signal is the interior of a circle of some radius r_1*. The following example considers a sequence that is nonzero for $-\infty < n < \infty$.

Example 3.1.5

Determine the z-transform of the signal

$$x(n) = \alpha^n u(n) + b^n u(-n-1)$$

Solution From definition (3.1.1) we have

$$X(z) = \sum_{n=0}^{\infty} \alpha^n z^{-n} + \sum_{n=-\infty}^{-1} b^n z^{-n} = \sum_{n=0}^{\infty} (\alpha z^{-1})^n + \sum_{l=1}^{\infty} (b^{-1}z)^l$$

The first power series converges if $|\alpha z^{-1}| < 1$ or $|z| > |\alpha|$. The second power series converges if $|b^{-1}z| < 1$ or $|z| < |b|$.

In determining the convergence of $X(z)$, we consider two different cases.

Case 1 $|b| < |\alpha|$: In this case the two ROC above do not overlap, as shown in Fig. 3.4(a). Consequently, we cannot find values of z for which both power series converge simultaneously. Clearly, in this case, $X(z)$ does not exist.

Case 2 $|b| > |\alpha|$: In this case there is a ring in the z-plane where both power series converge simultaneously, as shown in Fig. 3.4(b). Then we obtain

$$\begin{aligned} X(z) &= \frac{1}{1-\alpha z^{-1}} - \frac{1}{1-bz^{-1}} \\ &= \frac{b-\alpha}{\alpha + b - z - \alpha b z^{-1}} \end{aligned}$$

The ROC of $X(z)$ is $|\alpha| < |z| < |b|$.

This example shows that *if there is a ROC for an infinite duration two-sided signal, it is a ring (annular region) in the z-plane.* From Examples 3.1.1, 3.1.3, 3.1.4, and 3.1.5, we see that the ROC of a signal depends on both its duration (finite or infinite) and on whether it is causal, anticausal, or two-sided. These facts are summarized in Table 3.1.

One special case of a two-sided signal is a signal that has infinite duration on the right side but not on the left [i.e., $x(n) = 0$ for $n < n_0 < 0$]. A second case is a signal that has infinite duration on the left side but not on the

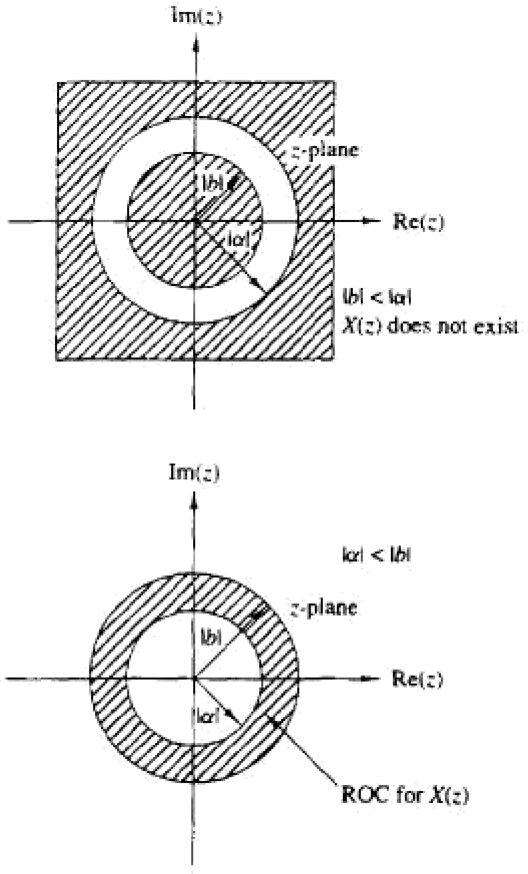


Figure 3.4 ROC for z-transform in Example 3.1.5.

3.1.2 The Inverse z-Transform

Often, we have the z-transform $X(z)$ of a signal and we must determine the signal sequence. The procedure for transforming from the z-domain to the time domain is called the *inverse z-transform*. An inversion formula for obtaining $x(n)$ from $X(z)$ can be derived by using the *Cauchy integral theorem*, which is an important theorem in the theory of complex variables.

To begin, we have the z-transform defined by (3.1.1) as

$$X(z) = \sum_{k=-\infty}^{\infty} x(k) z^{-k}$$

Suppose that we multiply both sides of (3.1.12) by z^{n-1} and integrate both sides over a closed contour within the ROC of $X(z)$ which encloses the origin. Such a contour is illustrated in Fig. 3.5. Thus we have

$$\oint_C X(z) z^{n-1} dz = \oint_C \sum_{k=-\infty}^{\infty} x(k) z^{n-1-k} dz$$

where C denotes the closed contour in the ROC of $X(z)$, taken in a counterclockwise direction. Since the series converges on this contour, we can interchange the order of integration and summation on the right-hand side of (3.1.13). Thus

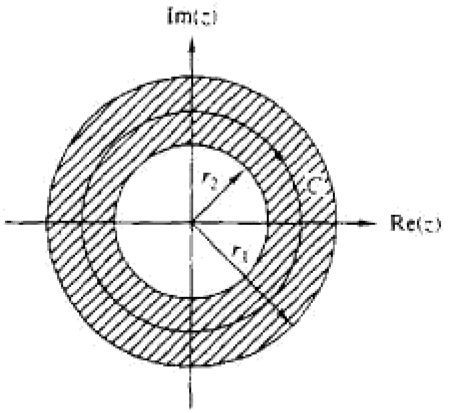


Figure 3.5 Contour C for integral in (3.1.13).

(3.1.13) becomes

$$\oint_C X(z)z^{n-1}dz = \sum_{k=-\infty}^{\infty} x(k)\oint_C z^{n-1-k}dz$$

Now we can invoke the Cauchy integral theorem, which states that

$$\frac{1}{2\pi j}\oint_C z^{n-1-k}\, dz = \begin{cases} 1, & k = n \\ 0, & k \neq n \end{cases}$$

where C is any contour that encloses the origin. By applying (3.1.15), the right-hand side of (3.1.14) reduces to $2\pi j x(n)$ and hence the desired inversion formula

$$x(n) = \frac{1}{2\pi j}\oint_C X(z)z^{n-1}\, dz$$

Although the contour integral in (3.1.16) provides the desired inversion formula for determining the sequence $x(n)$ from the z-transform, we shall not use (3.1.16) directly in our evaluation of inverse z-transforms. In our treatment we deal with signals and systems in the z-domain which have rational z-transforms (i.e., z-transforms that are a ratio of two polynomials). For such z-transforms we develop a simpler method for inversion that stems from (3.1.16) and employs a table lookup.

3.2 PROPERTIES OF THE Z-TRANSFORM

The z-transform is a very powerful tool for the study of discrete-time signals and systems. The power of this transform is a consequence of some very important properties that the transform possesses. In this section we examine some of these properties.

In the treatment that follows, it should be remembered that when we combine several z-transforms, the ROC of the overall transform is, at least, the intersection of the ROC of the individual transforms. This will become more apparent later, when we discuss specific examples.

Linearity. If

$$x_1(n) \overset{z}{\longleftrightarrow} X_1(z)$$

and

$$x_2(n) \overset{z}{\longleftrightarrow} X_2(z)$$

then

$$x(n) = a_1x_1(n) + a_2x_2(n) \overset{z}{\longleftrightarrow} X(z) = a_1X_1(z) + a_2X_2(z)$$

for any constants a_1 and a_2. The proof of this property follows immediately from the definition of linearity and is left as an exercise for the reader.

The linearity property can easily be generalized for an arbitrary number of signals. Basically, it implies that the z-transform of a linear combination of signals is the same linear combination of their z-transforms. Thus the linearity property helps us to find the z-transform of a signal by expressing the signal as a sum of elementary signals, for each of which, the z-transform is already known.

Example 3.2.1

Determine the z-transform and the ROC of the signal

$$x(n) = [3(2^n) - 4(3^n)]u(n)$$

Solution If we define the signals

$$x_1(n) = 2^n u(n)$$

and

$$x_2(n) = 3^n u(n)$$

then $x(n)$ can be written as

$$x(n) = 3x_1(n) - 4x_2(n)$$

According to (3.2.1), its z-transform is

$$X(z) = 3X_1(z) - 4X_2(z)$$

From (3.1.7) we recall that

$$\alpha^n u(n) \overset{z}{\longleftrightarrow} \frac{1}{1 - \alpha z^{-1}} \qquad \text{ROC: } |z| > |\alpha|$$

By setting $\alpha = 2$ and $\alpha = 3$ in (3.2.2), we obtain

$$x_1(n) = 2^n u(n) \overset{z}{\longleftrightarrow} X_1(z) = \frac{1}{1 - 2z^{-1}} \qquad \text{ROC: } |z| > 2$$

$$x_2(n) = 3^n u(n) \overset{z}{\longleftrightarrow} X_2(z) = \frac{1}{1 - 3z^{-1}} \qquad \text{ROC: } |z| > 3$$

The intersection of the ROC of $X_1(z)$ and $X_2(z)$ is $|z| > 3$. Thus the overall transform $X(z)$ is

$$X(z) = \frac{3}{1 - 2z^{-1}} - \frac{4}{1 - 3z^{-1}} \qquad \text{ROC: } |z| > 3$$

Example 3.2.2

Determine the z-transform of the signals

(a) $x(n) = (\cos \omega_0 n)u(n)$

(b) $x(n) = (\sin \omega_0 n)u(n)$

Solution

(a) By using Euler's identity, the signal $x(n)$ can be expressed as

$$x(n) = (\cos \omega_0 n)u(n) = \tfrac{1}{2}e^{j\omega_0 n}u(n) + \tfrac{1}{2}e^{-j\omega_0 n}u(n)$$

Thus (3.2.1) implies that

$$X(z) = \tfrac{1}{2}Z\{e^{j\omega_0 n}u(n)\} + \tfrac{1}{2}Z\{e^{-j\omega_0 n}u(n)\}$$

If we set $\alpha = e^{\pm j\omega_0}$ ($|\alpha| = |e^{\pm j\omega_0}| = 1$) in (3.2.2), we obtain

$$e^{j\omega_0 n}u(n) \overset{z}{\longleftrightarrow} \frac{1}{1 - e^{j\omega_0}z^{-1}} \qquad \text{ROC: } |z| > 1$$

and

$$e^{-j\omega_0 n}u(n) \overset{z}{\longleftrightarrow} \frac{1}{1 - e^{-j\omega_0}z^{-1}} \qquad \text{ROC: } |z| > 1$$

Thus

$$X(z) = \frac{1}{2}\frac{1}{1 - e^{j\omega_0}z^{-1}} + \frac{1}{2}\frac{1}{1 - e^{-j\omega_0}z^{-1}} \qquad \text{ROC: } |z| > 1$$

After some simple algebraic manipulations we obtain the desired result, namely,

$$(\cos\omega_0 n)u(n) \overset{z}{\longleftrightarrow} \frac{1 - z^{-1}\cos\omega_0}{1 - 2z^{-1}\cos\omega_0 + z^{-2}} \qquad \text{ROC: } |z| > 1 \tag{3.2.3}$$

(b) From Euler's identity,

$$x(n) = (\sin\omega_0 n)u(n) = \frac{1}{2j}[e^{j\omega_0 n}u(n) - e^{-j\omega_0 n}u(n)]$$

Thus

$$X(z) = \frac{1}{2j}\left(\frac{1}{1 - e^{j\omega_0}z^{-1}} - \frac{1}{1 - e^{-j\omega_0}z^{-1}}\right) \qquad \text{ROC: } |z| > 1$$

and finally,

$$(\sin\omega_0 n)u(n) \overset{z}{\longleftrightarrow} \frac{z^{-1}\sin\omega_0}{1 - 2z^{-1}\cos\omega_0 + z^{-2}} \qquad \text{ROC: } |z| > 1 \tag{3.2.4}$$

Time shifting. If

$$x(n) \overset{z}{\longleftrightarrow} X(z)$$

then

$$x(n-k) \overset{z}{\longleftrightarrow} z^{-k}X(z) \tag{3.2.5}$$

The ROC of $z^{-k}X(z)$ is the same as that of $X(z)$ except for $z = 0$ if $k > 0$ and $z = \infty$ if $k < 0$. The proof of this property follows immediately from the definition of the z-transform given in (3.1.1)

The properties of linearity and time shifting are the key features that make the z-transform extremely useful for the analysis of discrete-time LTI systems.

Example 3.2.3

By applying the time-shifting property, determine the z-transform of the signals $x_2(n)$ and $x_3(n)$ in Example 3.1.1 from the z-transform of $x_1(n)$.

Solution It can easily be seen that

$$x_2(n) = x_1(n+2)$$

and

$$x_3(n) = x_1(n-2)$$

Thus from (3.2.5) we obtain

$$X_2(z) = z^2 X_1(z) = z^2 + 2z + 5 + 7z^{-1} + z^{-3}$$

and

$$X_3(z) = z^{-2} X_1(z) = z^{-2} + 2z^{-3} + 5z^{-4} + 7z^{-5} + z^{-7}$$

Note that because of the multiplication by z^2, the ROC of $X_2(z)$ does not include the point $z = \infty$, even if it is contained in the ROC of $X_1(z)$.

Example 3.2.3 provides additional insight in understanding the meaning of the shifting property. Indeed, if we recall that the coefficient of z^{-n} is the sample value at time n, it is immediately seen that delaying a signal by $k (k > 0)$ samples [i.e., $x(n) \to x(n-k)$] corresponds to multiplying all terms of the z-transform by z^{-k}. The coefficient of z^{-n} becomes the coefficient of $z^{-(n+k)}$.

Example 3.2.4

Determine the transform of the signal

$$x(n) = \begin{cases} 1, & 0 \le n \le N-1 \\ 0, & \text{elsewhere} \end{cases}$$

Solution We can determine the z-transform of this signal by using the definition (3.1.1). Indeed,

$$X(z) = \sum_{n=0}^{N-1} 1 \cdot z^{-n} = 1 + z^{-1} + \cdots + z^{-(N-1)} = \begin{cases} N, & \text{if } z = 1 \\ \dfrac{1 - z^{-N}}{1 - z^{-1}}, & \text{if } z \neq 1 \end{cases}$$

Since $x(n)$ has finite duration, its ROC is the entire z-plane, except $z = 0$.

Let us also derive this transform by using the linearity and time shifting properties. Note that $x(n)$ can be expressed in terms of two unit step signals

$$x(n) = u(n) - u(n-N)$$

By using (3.2.1) and (3.2.5) we have

$$X(z) = Z\{u(n)\} - Z\{u(n-N)\} = (1 - z^{-N}) Z\{u(n)\}$$

However, from (3.1.8) we have

$$Z\{u(n)\} = \frac{1}{1 - z^{-1}} \qquad \text{ROC: } |z| > 1$$

which, when combined with (3.2.8), leads to (3.2.7).

Example 3.2.4 helps to clarify a very important issue regarding the ROC of the combination of several z-transforms. If the linear combination of several signals has finite duration, the ROC of its z-transform is exclusively dictated by the finite-duration nature of this signal, not by the ROC of the individual transforms.

Scaling in the z-domain. If

$$x(n) \xleftrightarrow{z} X(z) \qquad \text{ROC: } r_1 < |z| < r_2$$

then

$$a^n x(n) \overset{z}{\longleftrightarrow} X(a^{-1}z) \qquad \text{ROC: } |a|r_1 < |z| < |a|r_2$$

for any constant a, real or complex.

Proof. From the definition (3.1.1)

$$Z\{a^n x(n)\} = \sum_{n=-\infty}^{\infty} a^n x(n) z^{-n} = \sum_{n=-\infty}^{\infty} x(n)(a^{-1}z)^{-n}$$
$$= X(a^{-1}z)$$

Since the ROC of $X(z)$ is $r_1 < |z| < r_2$, the ROC of $X(a^{-1}z)$ is

$$r_1 < |a^{-1}z| < r_2$$

or

$$|a|r_1 < |z| < |a|r_2$$

To better understand the meaning and implications of the scaling property, we express a and z in polar form as $a = r_0 e^{j\omega_0}$, $z = re^{j\omega}$, and we introduce a new complex variable $w = a^{-1}z$. Thus $Z\{x(n)\} = X(z)$ and $Z\{a^n x(n)\} = X(w)$. It can easily be seen that

$$w = a^{-1}z = \left(\frac{1}{r_0} r\right) e^{j(\omega-\omega_0)}$$

This change of variables results in either shrinking (if $r_0 > 1$) or expanding (if $r_0 < 1$) the z-plane in combination with a rotation (if $\omega_0 \neq 2k\pi$) of the z-plane (see Fig. 3.6). This explains why we have a change in the ROC of the new transform where $|a| < 1$. The case $|a| = 1$, that is, $a = e^{j\omega_0}$ is of special interest because it corresponds only to rotation of the z-plane.

Example 3.2.5

Determine the z-transforms of the signals

(a) $x(n) = a^n(\cos \omega_0 n)u(n)$

(b) $x(n) = a^n(\sin \omega_0 n)u(n)$

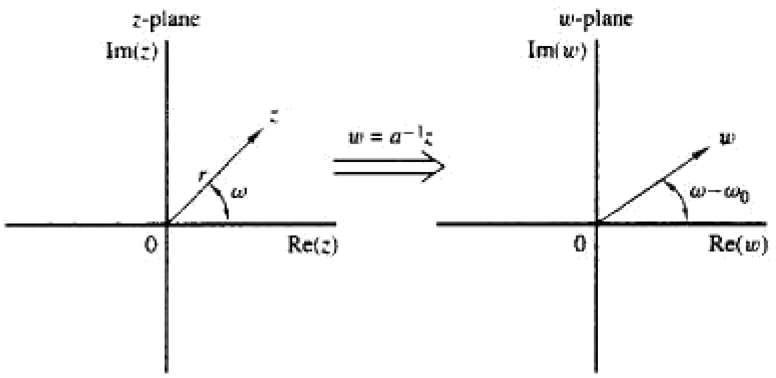


Figure 3.6 Mapping of the z-plane to the w-plane via the transformation $w = a^{-1}z$, $a = r_0 e^{j\omega_0}$.

Solution

(a) From (3.2.3) and (3.2.9) we easily obtain

$$a^n(\cos\omega_0 n)u(n) \overset{z}{\longleftrightarrow} \frac{1 - az^{-1}\cos\omega_0}{1 - 2az^{-1}\cos\omega_0 + a^2z^{-2}} \qquad |z| > |a| \qquad (3.2.10)$$

(b) Similarly, (3.2.4) and (3.2.9) yield

$$a^n(\sin\omega_0 n)u(n) \overset{z}{\longleftrightarrow} \frac{az^{-1}\sin\omega_0}{1 - 2az^{-1}\cos\omega_0 + a^2z^{-2}} \qquad |z| > |a| \qquad (3.2.11)$$

Time reversal. If

$$x(n) \overset{z}{\longleftrightarrow} X(z) \qquad \text{ROC: } r_1 < |z| < r_2$$

then

$$x(-n) \overset{z}{\longleftrightarrow} X(z^{-1}) \qquad \text{ROC: } \frac{1}{r_2} < |z| < \frac{1}{r_1} \qquad (3.2.12)$$

Proof. From the definition (3.1.1), we have

$$Z\{x(-n)\} = \sum_{n=-\infty}^{\infty} x(-n)z^{-n} = \sum_{l=-\infty}^{\infty} x(l)(z^{-1})^{-l} = X(z^{-1})$$

where the change of variable $l = -n$ is made. The ROC of $X(z^{-1})$ is

$$r_1 < |z^{-1}| < r_2 \quad \text{or equivalently} \quad \frac{1}{r_2} < |z| < \frac{1}{r_1}$$

Note that the ROC for $x(n)$ is the inverse of that for $x(-n)$. This means that if z_0 belongs to the ROC of $x(n)$, then $1/z_0$ is in the ROC for $x(-n)$.

An intuitive proof of (3.2.12) is the following. When we fold a signal, the coefficient of z^{-n} becomes the coefficient of z^n. Thus, folding a signal is equivalent to replacing z by z^{-1} in the z-transform formula. In other words, reflection in the time domain corresponds to inversion in the z-domain.

TABLE 3.2 PROPERTIES OF THE *Z*-TRANSFORM

Property	Time Domain	z-Domain	ROC
Notation	$x(n)$ $x_1(n)$ $x_2(n)$	$X(z)$ $X_1(z)$ $X_2(z)$	ROC: $r_2 < \|z\| < r_1$ ROC$_1$ ROC$_2$
Linearity	$a_1x_1(n) + a_2x_2(n)$	$a_1X_1(z) + a_2X_2(z)$	At least the intersection of ROC$_1$ and ROC$_2$
Time shifting	$x(n-k)$	$z^{-k}X(z)$	That of $X(z)$, except $z = 0$ if $k > 0$ and $z = \infty$ if $k < 0$
Scaling in the z-domain	$a^n x(n)$	$X(a^{-1}z)$	$\|a\|r_2 < \|z\| < \|a\|r_1$
Time reversal	$x(-n)$	$X(z^{-1})$	$\frac{1}{r_1} < \|z\| < \frac{1}{r_2}$
Conjugation	$x^*(n)$	$X^*(z^*)$	ROC
Real part	$\mathrm{Re}\{x(n)\}$	$\frac{1}{2}[X(z) + X^*(z^*)]$	Includes ROC
Imaginary part	$\mathrm{Im}\{x(n)\}$	$\frac{1}{2}[X(z) - X^*(z^*)]$	Includes ROC
Differentiation in the z-domain	$nx(n)$	$-z\frac{dX(z)}{dz}$	$r_2 < \|z\| < r_1$
Convolution	$x_1(n) * x_2(n)$	$X_1(z)X_2(z)$	At least, the intersection of ROC$_1$ and ROC$_2$
Correlation	$r_{x_1x_2}(l) = x_1(l) * x_2(-l)$	$R_{x_1x_2}(z) = X_1(z)X_2(z^{-1})$	At least, the intersection of ROC of $X_1(z)$ and $X_2(z^{-1})$
Initial value theorem	If $x(n)$ causal	$x(0) = \lim_{z\to\infty} X(z)$	
Multiplication	$x_1(n)x_2(n)$	$\frac{1}{2\pi j}\oint_C X_1(v)X_2\left(\frac{z}{v}\right)v^{-1}dv$	At least $r_{1l}r_{2l} < \|z\| < r_{1u}r_{2u}$
Parseval's relation	$\sum_{n=-\infty}^{\infty} x_1(n)x_2^*(n) = \frac{1}{2\pi j}\oint_C X_1(v)X_2^*(1/v^*)v^{-1}dv$		

Example 3.3.1

Determine the pole–zero plot for the signal

$$x(n) = a^n u(n) \qquad a > 0$$

Solution From Table 3.3 we find that

$$X(z) = \frac{1}{1 - az^{-1}} = \frac{z}{z - a} \qquad \text{ROC: } |z| > a$$

Thus $X(z)$ has one zero at $z_1 = 0$ and one pole at $p_1 = a$. The pole–zero plot is shown in Fig. 3.7. Note that the pole $p_1 = a$ is not included in the ROC since the z-transform does not converge at a pole.

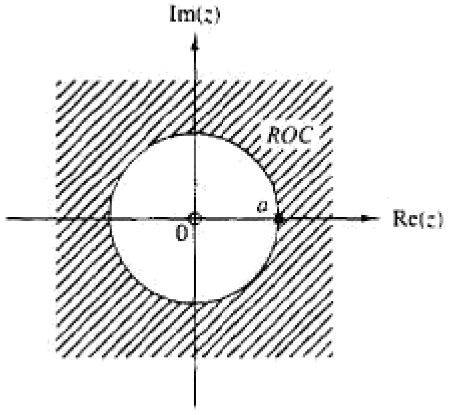


Figure 3.7 Pole–zero plot for the causal exponential signal $x(n) = a^n u(n)$.

Example 3.3.2

Determine the pole–zero plot for the signal

$$x(n) = \begin{cases} a^n, & 0 \le n \le M - 1 \\ 0, & \text{elsewhere} \end{cases}$$

where $a > 0$.

Solution From the definition (3.1.1) we obtain

$$X(z) = \sum_{n=0}^{M-1} (az^{-1})^n = \frac{1 - (az^{-1})^M}{1 - az^{-1}} = \frac{z^M - a^M}{z^{M-1}(z - a)}$$

Since $a > 0$, the equation $z^M = a^M$ has M roots at

$$z_k = ae^{j2\pi k/M} \qquad k = 0, 1, \ldots, M - 1$$

The zero $z_0 = a$ cancels the pole at $z = a$. Thus

$$X(z) = \frac{(z - z_1)(z - z_2)\cdots(z - z_{M-1})}{z^{M-1}}$$

which has $M - 1$ zeros and $M - 1$ poles, located as shown in Fig. 3.8 for $M = 8$. Note that the ROC is the entire z-plane except $z = 0$ because of the $M - 1$ poles located at the origin.

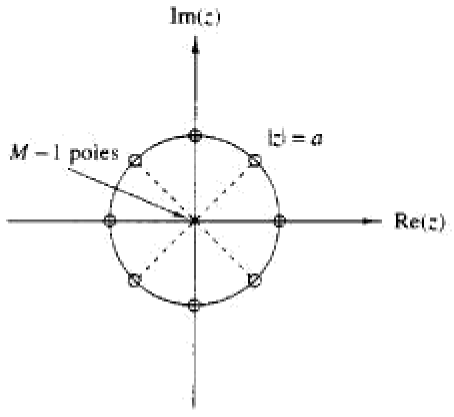


Figure 3.8 Pole–zero pattern for the finite-duration signal $x(n) = a^n$, $0 \le n \le M - 1 (a > 0)$, for $M = 8$.

Clearly, if we are given a pole–zero plot, we can determine $X(z)$, by using (3.3.2), to within a scaling factor G. This is illustrated in the following example.

Example 3.3.3

Determine the z-transform and the signal that corresponds to the pole–zero plot of Fig. 3.9.

Solution There are two zeros ($M = 2$) at $z_1 = 0$, $z_2 = r\cos\omega_0$ and two poles ($N = 2$) at $p_1 = re^{j\omega_0}$, $p_2 = re^{-j\omega_0}$. By substitution of these relations into (3.3.2), we obtain

$$X(z) = G\frac{(z - z_1)(z - z_2)}{(z - p_1)(z - p_2)} = G\frac{z(z - r\cos\omega_0)}{(z - re^{j\omega_0})(z - re^{-j\omega_0})} \qquad \text{ROC: } |z| > r$$

After some simple algebraic manipulations, we obtain

$$X(z) = G\frac{1 - rz^{-1}\cos\omega_0}{1 - 2rz^{-1}\cos\omega_0 + r^2z^{-2}} \qquad \text{ROC: } |z| > r$$

From Table 3.3 we find that

$$x(n) = G(r^n\cos\omega_0 n)u(n)$$

From Example 3.3.3, we see that the product $(z - p_1)(z - p_2)$ results in a polynomial with real coefficients, when p_1 and p_2 are complex conjugates. In

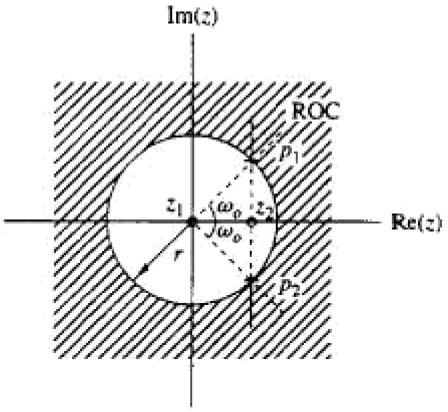


Figure 3.9 Pole-zero pattern for Example 3.3.3.

Analysis of discrete time system using Ƶ-transform

The convolution property of Ƶ-transform says that the Ƶ-transform of the convolution of x(n) and h(n) is equal to the product of their individual Ƶ-transforms.

$$\therefore \mathcal{Z}\{x(n) * h(n)\} = X(z)\, H(z)$$

From eqn(1.29) we know that,

$$y(n) = x(n) * h(n)$$

Using equation(1.34) in eqn(1.33) we get

$$\mathcal{Z}\{y(n)\} = X(z)\, H(z)$$

$$\therefore Y(z) = X(z)H(z)$$

where $Y(z) = \mathcal{Z}\{y(n)\}$

Now the response y(n) of LTI system is obtained by taking inverse Ƶ-transform of equation(1.35)

$$\therefore y(n) = \mathcal{Z}^{-1}\{X(z)H(z)\}$$

Hence in order to determine the response of LTI system, first determine the Ƶ-transform of the input signal and impulse response of the system.

Let X(z) = Ƶ-transform of input signal x(n) and

H(z) = Ƶ-transform of impulse response h(n).

Now take the product of X(z) and H(z), then determine the inverse $\mathcal{Z}$-transform of the product, which gives the response y(n) of the system.

From equation (1.35) we can write,

$$H(z) = \frac{Y(z)}{X(z)}$$

We know that Y(z)/X(z) is the transfer function of LTI system. Therefore from equation (1.37) we can say that the transfer function of LTI system is also given by the $\mathcal{Z}$-transform of impulse response.

Conversely we can say that if the transfer function of the system is known then we can determine the impulse response of the system by taking inverse $\mathcal{Z}$-transform of the transfer function

$$\therefore \text{Impulse response, } h(n) = \mathcal{Z}^{-1}\{H(z)\} = \mathcal{Z}^{-1}\left\{\frac{Y(z)}{X(z)}\right\}$$

EXAMPLE 1.1

Determine the impulse response h(n) for the system described by the second order difference equation, $y(n) - 4y(n-1) + 4y(n-2) = x(n-1)$.

SOLUTION

The difference equation governing the system is

$$y(n) - 4y(n-1) + 4y(n-2) = x(n-1)$$

Let $\mathcal{Z}\{y(n)\} = Y(z)$, $\therefore \mathcal{Z}\{a\,y(n-k)\} = a\,z^{-k}Y(z)$

Let $\mathcal{Z}\{x(n)\} = X(z)$ $\therefore \mathcal{Z}\{a\,x(n-k)\} = a\,z^{-k}X(z)$

where a is constant

On taking $\mathcal{Z}$-transform of the difference equation governing the system we get,

$$Y(z) - 4z^{-1}Y(z) + 4z^{-2}Y(z) = z^{-1}X(z)$$

$$(1 - 4z^{-1} + 4z^{-2})\,Y(z) = z^{-1}X(z)$$

$$\therefore \frac{Y(z)}{X(z)} = \frac{z^{-1}}{1-4z^{-1}+4z^{-2}}$$

We know that, $\dfrac{Y(z)}{X(z)} = H(z)$

$$\therefore H(z) = \frac{z^{-1}}{1-4z^{-1}+4z^{-2}} = \frac{z^{-1}}{z^{-2}(z^2-4z+4)} = \frac{z}{(z-2)^2} = \frac{1}{2}\frac{2z}{(z-2)^2}$$

Impulse response, h(n) $= \mathcal{Z}^{-1}\{H(z)\} = \mathcal{Z}^{-1}\left\{\frac{1}{2}\frac{2z}{(z-2)^2}\right\}$

$= (1/2)\, n2^n = n\, 2^{(n-1)}$ for $n \geq 0$

or $h(n) = n2^{(n-1)}\, u(n)$ for all n.

We know that

$$\mathcal{Z}\{na^n\} = \frac{az}{(z-a)^2}$$

EXAMPLE 1.2

Find the transfer function and unit sample response of the second order difference equation with zero initial condition, $y(nT) = x(nT) - 0.25y(nT-2T)$

SOLUTION

The difference equation governing the system is

$$y(nT) = x(nT) - 0.25\, y(nT-2T)$$

Let $\mathcal{Z}\{y(nT)\} = Y(z)$,

$\therefore \mathcal{Z}\{ay(nT-2T)\} = a\, z^{-2}\, Y(z)$ when the initial conditions are zero.

Let $\mathcal{Z}\{x(nT)\} = X(z)$

On taking $\mathcal{Z}$-transform of the difference equation governing the system we get

$$Y(z) = X(z) - 0.25\, z^{-2}\, Y(z)$$

$$Y(z) + 0.25z^{-2}\, Y(z) = X(z)$$

$$(1+0.25z^{-2})\, Y(z) = X(z)$$

$$\therefore \frac{Y(z)}{X(z)} = \frac{1}{1+0.25z^{-2}}$$

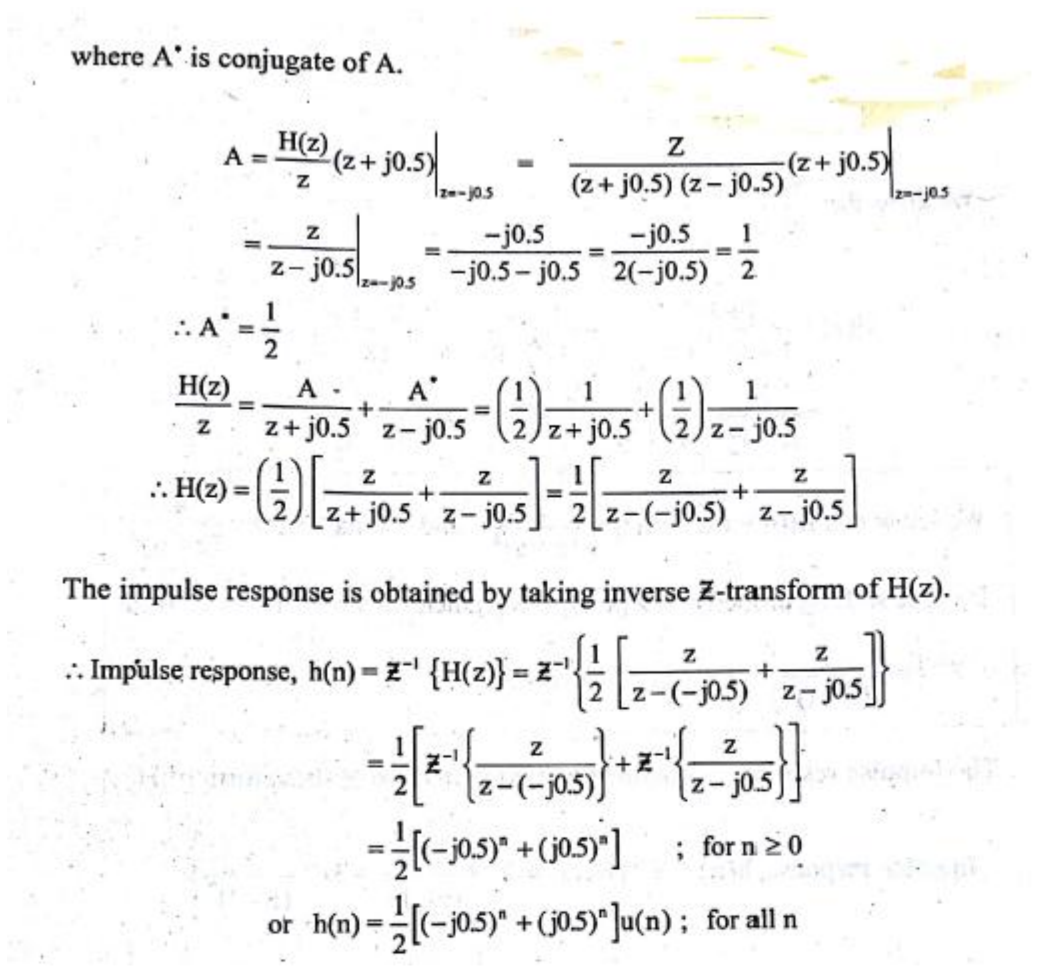
The equation (1.2.2) is the transfer function of the system.

We know that, $\frac{Y(z)}{X(z)} = H(z)$

$$\therefore H(z) = \frac{1}{1+0.25\, z^{-2}} = \frac{1}{z^{-2}(z^2+0.25)} = \frac{z^2}{(z+j\,0.5)\,(z-j\,0.5)}$$

By partial fraction expansion we can write,

$$\frac{H(z)}{z} = \frac{z}{(z+j0.5)\,(z-j0.5)} = \frac{A}{z+j0.5} + \frac{A^*}{z-j0.5}$$


where A^* is conjugate of A.

$$A = \frac{H(z)}{z}(z+j0.5)\Big|_{z=-j0.5} = \frac{z}{(z+j0.5)(z-j0.5)}(z+j0.5)\Big|_{z=-j0.5}$$

$$= \frac{z}{z-j0.5}\Big|_{z=-j0.5} = \frac{-j0.5}{-j0.5-j0.5} = \frac{-j0.5}{2(-j0.5)} = \frac{1}{2}$$

$$\therefore A^* = \frac{1}{2}$$

$$\frac{H(z)}{z} = \frac{A}{z+j0.5} + \frac{A^*}{z-j0.5} = \left(\frac{1}{2}\right)\frac{1}{z+j0.5} + \left(\frac{1}{2}\right)\frac{1}{z-j0.5}$$

$$\therefore H(z) = \left(\frac{1}{2}\right)\left[\frac{z}{z+j0.5} + \frac{z}{z-j0.5}\right] = \frac{1}{2}\left[\frac{z}{z-(-j0.5)} + \frac{z}{z-j0.5}\right]$$

The impulse response is obtained by taking inverse $\mathcal{Z}$-transform of H(z).

$$\therefore \text{Impulse response, } h(n) = \mathcal{Z}^{-1}\{H(z)\} = \mathcal{Z}^{-1}\left\{\frac{1}{2}\left[\frac{z}{z-(-j0.5)} + \frac{z}{z-j0.5}\right]\right\}$$

$$= \frac{1}{2}\left[\mathcal{Z}^{-1}\left\{\frac{z}{z-(-j0.5)}\right\} + \mathcal{Z}^{-1}\left\{\frac{z}{z-j0.5}\right\}\right]$$

$$= \frac{1}{2}\left[(-j0.5)^n + (j0.5)^n\right] \quad ; \text{ for } n \geq 0$$

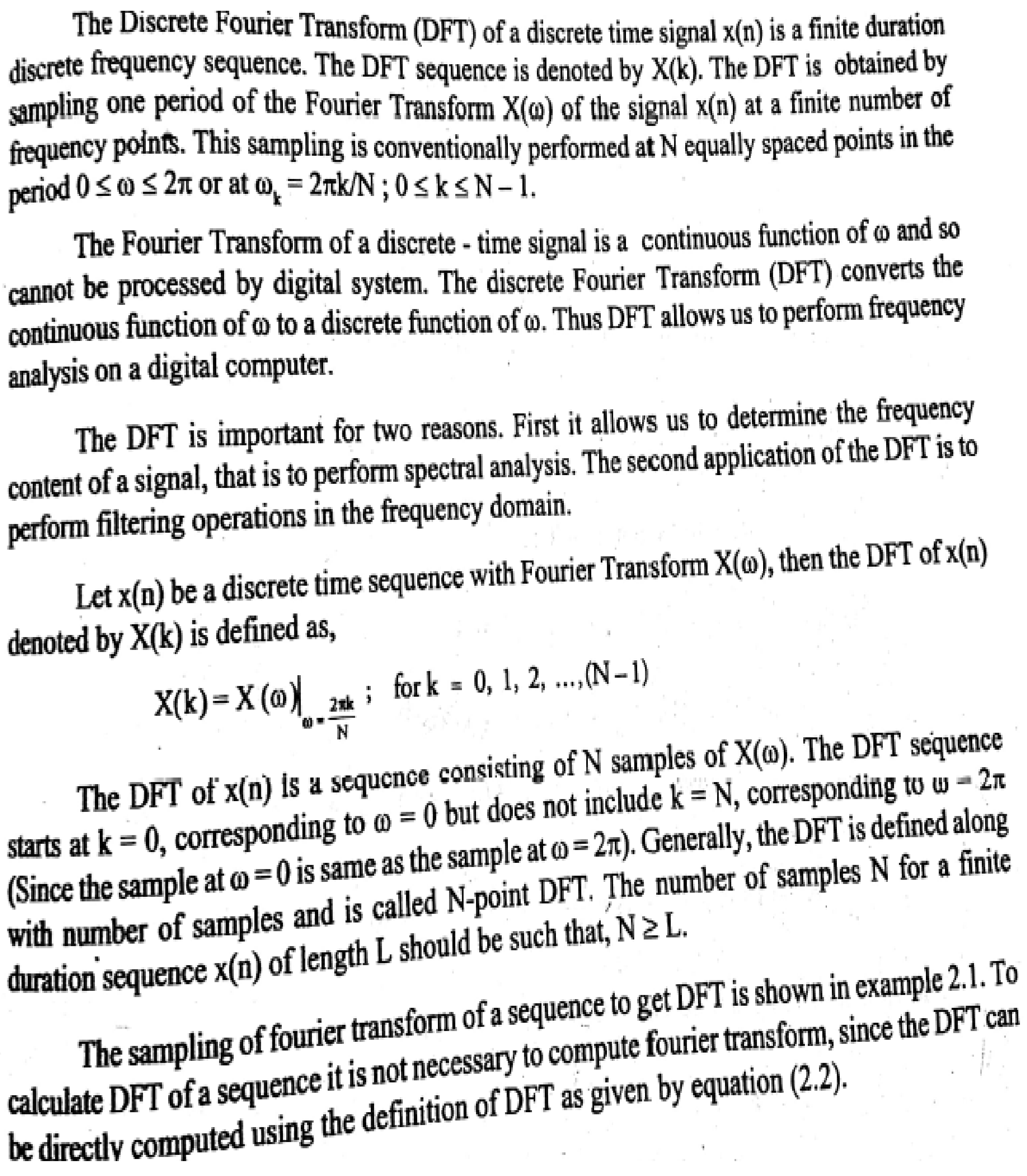
$$\text{or } h(n) = \frac{1}{2}\left[(-j0.5)^n + (j0.5)^n\right]u(n) \; ; \text{ for all } n$$

FREQUENCY ANALYSIS OF SIGNALS

The discrete Fourier transform (DFT) converts a finite sequence of equally-spaced samples of a function into a same-length sequence of equally-spaced samples of the discrete-time Fourier transform (DTFT), which is a complex-valued function of frequency. The interval at which the DTFT is sampled is the reciprocal of the duration of the input sequence. An inverse DFT is a Fourier series, using the DTFT samples as coefficients of complexsinusoids at the corresponding DTFT frequencies. It has the same sample-values as the original input sequence. The DFT is therefore said to be a frequency domain representation of the original input sequence. If the original sequence spans all the non-zero values of a function, its DTFT is continuous (and periodic), and the DFT provides discrete samples of one cycle. If the original sequence is one cycle of a periodic function, the DFT provides all the non-zero values of one DTFT cycle.

The DFT is the most important discrete transform, used to perform Fourier analysis in many practical applications.[1] In digital signal processing, the function is any quantity or signal that varies over time, such as the pressure of a sound wave, a radio signal, or daily temperature readings, sampled over a finite time interval (often defined by a window function[2]). In image processing, the samples can be the values of pixels along a row or column of a raster image. The DFT is also used to efficiently solve partial differential equations, and to perform other operations such as convolutions or multiplying large integers.

Since it deals with a finite amount of data, it can be implemented in computers by numerical algorithms or even dedicated hardware. These implementations usually employ efficient fast Fourier transform (FFT) algorithms;[3] so much so that the terms "FFT" and "DFT" are often used interchangeably. Prior to its current usage, the "FFT" initialism may have also been used for the ambiguous term "finite Fourier transform".



The Discrete Fourier Transform (DFT) of a discrete time signal x(n) is a finite duration discrete frequency sequence. The DFT sequence is denoted by X(k). The DFT is obtained by sampling one period of the Fourier Transform X(ω) of the signal x(n) at a finite number of frequency points. This sampling is conventionally performed at N equally spaced points in the period $0 \le \omega \le 2\pi$ or at $\omega_k = 2\pi k/N$; $0 \le k \le N-1$.

The Fourier Transform of a discrete - time signal is a continuous function of ω and so cannot be processed by digital system. The discrete Fourier Transform (DFT) converts the continuous function of ω to a discrete function of ω. Thus DFT allows us to perform frequency analysis on a digital computer.

The DFT is important for two reasons. First it allows us to determine the frequency content of a signal, that is to perform spectral analysis. The second application of the DFT is to perform filtering operations in the frequency domain.

Let x(n) be a discrete time sequence with Fourier Transform X(ω), then the DFT of x(n) denoted by X(k) is defined as,

$$X(k) = X(\omega)\Big|_{\omega = \frac{2\pi k}{N}} \; ; \quad \text{for } k = 0, 1, 2, \ldots, (N-1)$$

The DFT of x(n) is a sequence consisting of N samples of X(ω). The DFT sequence starts at k = 0, corresponding to ω = 0 but does not include k = N, corresponding to ω = 2π (Since the sample at ω = 0 is same as the sample at ω = 2π). Generally, the DFT is defined along with number of samples and is called N-point DFT. The number of samples N for a finite duration sequence x(n) of length L should be such that, N ≥ L.

The sampling of fourier transform of a sequence to get DFT is shown in example 2.1. To calculate DFT of a sequence it is not necessary to compute fourier transform, since the DFT can be directly computed using the definition of DFT as given by equation (2.2).

Definition of DFT

The N-point DFT of a finite duration sequence x(n) of length L, where N ≥ L, is defined as

$$\text{DFT}\{x(n)\} = X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j\frac{2\pi kn}{N}} \;;\; \text{for } k = 0,1,2,\ldots\ldots,(N-1)$$

Definition of IDFT

The Inverse Discrete Fourier Transform (IDFT) of the sequence X(k) of length N is defined as

$$\text{IDFT}\{X(k)\} = x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)\, e^{\frac{j2\pi kn}{N}} \;;\; \text{for } n = 0,1,\ldots\ldots,(N-1)$$

The equation (2.4) is used to denote the N-point DFT pair x(n) and X(k).

$$x(n) \xmapsto[N]{\text{DFT}} X(k)$$

EXAMPLE

Compute 4-point DFT of causal three sample sequence given by

$$x(n) = \frac{1}{3} \;;\; 0 \le n \le 2$$
$$= 0 \;;\; \text{else}$$

SOLUTION

By the definition of N-point DFT, the kth complex coefficient of X(k), for $0 \le k \le N-1$, is given by,

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi kn/N}$$

Here N = 4, therefore the 4-point DFT is

$$X(k) = \sum_{n=0}^{3} x(n)\, e^{-j2\pi kn/4} = \sum_{n=0}^{3} x(n)\, e^{-j\pi kn/2}$$

$$= x(0)e^{0} + x(1)e^{-j\pi k/2} + x(2)e^{-j\pi k} + x(3)e^{-j3\pi k/2}$$

$$= \frac{1}{3} + \frac{1}{3}e^{-j\pi k/2} + \frac{1}{3}e^{-j\pi k} + 0 = \frac{1}{3}\left[1 + e^{-j\pi k/2} + e^{-j\pi k}\right]$$

$$= \frac{1}{3}\left[1 + \cos\frac{\pi k}{2} - j\sin\frac{\pi k}{2} + \cos\pi k - j\sin\pi k\right]$$

The values of X(k) can be evaluated for k = 0, 1, 2, 3

When k = 0; $X(0) = \frac{1}{3}[1 + \cos 0 - j\sin 0 + \cos 0 - j\sin 0]$

$$= \frac{1}{3}(1+1+1) = 1 = 1\angle 0$$

When k = 1; $X(1) = \frac{1}{3}\left[1 + \cos\frac{\pi}{2} - j\sin\frac{\pi}{2} + \cos\pi - j\sin\pi\right]$

$$= \frac{1}{3}(1 + 0 - j - 1 - j0) = -j\frac{1}{3} = \frac{1}{3}\angle -\pi/2$$

When k = 2; $X(2) = \frac{1}{3}[1 + \cos\pi - j\sin\pi + \cos 2\pi - j\sin 2\pi]$

$$= \frac{1}{3}(1 - 1 - j0 + 1 - j0) = \frac{1}{3} = \frac{1}{3}\angle 0$$

When k = 3; $X(3) = \frac{1}{3}\left[1 + \cos\frac{3\pi}{2} - j\sin\frac{3\pi}{2} + \cos 3\pi - j\sin 3\pi\right]$

$$= \frac{1}{3}(1 + 0 + j - 1 - j0) = j\frac{1}{3} = \frac{1}{3}\angle\pi/2$$

$\therefore$ The 4-point DFT sequence of x(n) is given by,

$$X(k) = \{1\angle 0,\ \frac{1}{3}\angle -\pi/2,\ \frac{1}{3}\angle 0,\ \frac{1}{3}\angle\pi/2$$

Magnitude function, $|X(k)| = \{1,\ \frac{1}{3},\ \frac{1}{3},\ \frac{1}{3}\}$

Phase function, $\angle X(k) = \{0, -\pi/2, 0, \pi/2\}$

PROPERTIES OF DFT

Property	Time Domain	Frequency Domain
Notation	$x(n), y(n)$	$X(k), Y(k)$
Periodicity	$x(n) = x(n+N)$	$X(k) = X(k+N)$
Linearity	$a_1x_1(n) + a_2x_2(n)$	$a_1X_1(k) + a_2X_2(k)$
Time reversal	$x(N-n)$	$X(N-k)$
Circular time shift	$x((n-l))_N$	$X(k)e^{-j2\pi kl/N}$
Circular frequency shift	$x(n)e^{j2\pi ln/N}$	$X((k-l))_N$
Complex conjugate	$x^*(n)$	$X^*(N-k)$
Circular convolution	$x_1(n) \textcircled{N} x_2(n)$	$X_1(k)X_2(k)$
Circular correlation	$x(n) \textcircled{N} y^*(-n)$	$X(k)Y^*(k)$
Multiplication of two sequences	$x_1(n)x_2(n)$	$\frac{1}{N}X_1(k) \textcircled{N} X_2(k)$
Parseval's theorem	$\sum_{n=0}^{N-1} x(n)y^*(n)$	$\frac{1}{N}\sum_{k=0}^{N-1} X(k)Y^*(k)$

If we multiply the two DFTs together, the result is a DFT, say $X_3(k)$, of a sequence $x_3(n)$ of length N. Let us determine the relationship between $x_3(n)$ and the sequences $x_1(n)$ and $x_2(n)$.

We have

$$X_3(k) = X_1(k)X_2(k) \qquad k = 0, 1, \ldots, N-1$$

The IDFT of $\{X_3(k)\}$ is

$$\begin{aligned} x_3(m) &= \frac{1}{N}\sum_{k=0}^{N-1} X_3(k)e^{j2\pi km/N} \\ &= \frac{1}{N}\sum_{k=0}^{N-1} X_1(k)X_2(k)e^{j2\pi km/N} \end{aligned}$$

Suppose that we substitute for $X_1(k)$ and $X_2(k)$ in (5.2.35) using the DFTs given in (5.2.32) and (5.2.33). Thus we obtain

$$\begin{aligned} x_3(m) &= \frac{1}{N}\sum_{k=0}^{N-1}\left[\sum_{n=0}^{N-1} x_1(n)e^{-j2\pi kn/N}\right]\left[\sum_{l=0}^{N-1} x_2(l)e^{-j2\pi kl/N}\right]e^{j2\pi km/N} \\ &= \frac{1}{N}\sum_{n=0}^{N-1} x_1(n)\sum_{l=0}^{N-1} x_2(l)\left[\sum_{k=0}^{N-1} e^{j2\pi k(m-n-l)/N}\right] \end{aligned}$$

The inner sum in the brackets in (5.2.36) has the form

$$\sum_{k=0}^{N-1} a^k = \begin{cases} N, & a = 1 \\ \dfrac{1-a^N}{1-a}, & a \neq 1 \end{cases}$$

where a is defined as

$$a = e^{j2\pi(m-n-l)/N}$$

We observe that $a = 1$ when $m - n - l$ is a multiple of N. On the other hand, $a^N = 1$ for any value of $a \neq 0$. Consequently, (5.2.37) reduces to

$$\sum_{k=0}^{N-1} a^k = \begin{cases} N, & l = m - n + pN = ((m-n))_N, \quad p \text{ an integer} \\ 0, & \text{otherwise} \end{cases}$$

If we substitute the result in (5.2.38) into (5.2.36), we obtain the desired expression for $x_3(m)$ in the form

$$x_3(m) = \sum_{n=0}^{N-1} x_1(n)x_2((m-n))_N \qquad m = 0, 1, \ldots, N-1$$

The expression in (5.2.39) has the form of a convolution sum. However, it is not the ordinary linear convolution that was introduced in Chapter 2, which relates the output sequence $y(n)$ of a linear system to the input sequence $x(n)$ and the impulse response $h(n)$. Instead, the convolution sum in (5.2.39) involves the index

$((m-n))_N$ and is called *circular convolution*. Thus we conclude that multiplication of the DFTs of two sequences is equivalent to the circular convolution of the two sequences in the time domain.

The following example illustrates the operations involved in circular convolution.

Example 5.2.1

Perform the circular convolution of the following two sequences:

$$x_1(n) = \{\underset{\uparrow}{2}, 1, 2, 1\}$$

$$x_2(n) = \{\underset{\uparrow}{1}, 2, 3, 4\}$$

Solution Each sequence consists of four nonzero points. For the purposes of illustrating the operations involved in circular convolution, it is desirable to graph each sequence as points on a circle. Thus the sequences $x_1(n)$ and $x_2(n)$ are graphed as illustrated in Fig. 5.8(a). We note that the sequences are graphed in a counterclockwise direction on a circle. This establishes the reference direction in rotating one of the sequences relative to the other.

Now, $x_3(m)$ is obtained by circularly convolving $x_1(n)$ with $x_2(n)$ as specified by (5.2.39). Beginning with $m = 0$ we have

$$x_3(0) = \sum_{n=0}^{3} x_1(n)x_2((-n))_N$$

$x_2((-n))_4$ is simply the sequence $x_2(n)$ folded and graphed on a circle as illustrated in Fig. 5.8(b). In other words, the folded sequence is simply $x_2(n)$ graphed in a clockwise direction.

The product sequence is obtained by multiplying $x_1(n)$ with $x_2((-n))_4$, point by point. This sequence is also illustrated in Fig. 5.8(b). Finally, we sum the values in the product sequence to obtain

$$x_3(0) = 14$$

For $m = 1$ we have

$$x_3(1) = \sum_{n=0}^{3} x_1(n)x_2((1-n))_4$$

It is easily verified that $x_2((1-n))_4$ is simply the sequence $x_2((-n))_4$ rotated counterclockwise by one unit in time as illustrated in Fig. 5.8(c). This rotated sequence multiplies $x_1(n)$ to yield the product sequence, also illustrated in Fig. 5.8(c). Finally, we sum the values in the product sequence to obtain $x_3(1)$. Thus

$$x_3(1) = 16$$

For $m = 2$ we have

$$x_3(2) = \sum_{n=0}^{3} x_1(n)x_2((2-n))_4$$

Now $x_2((2-n))_4$ is the folded sequence in Fig. 5.8(b) rotated two units of time in the counterclockwise direction. The resultant sequence is illustrated in Fig. 5.8(d)

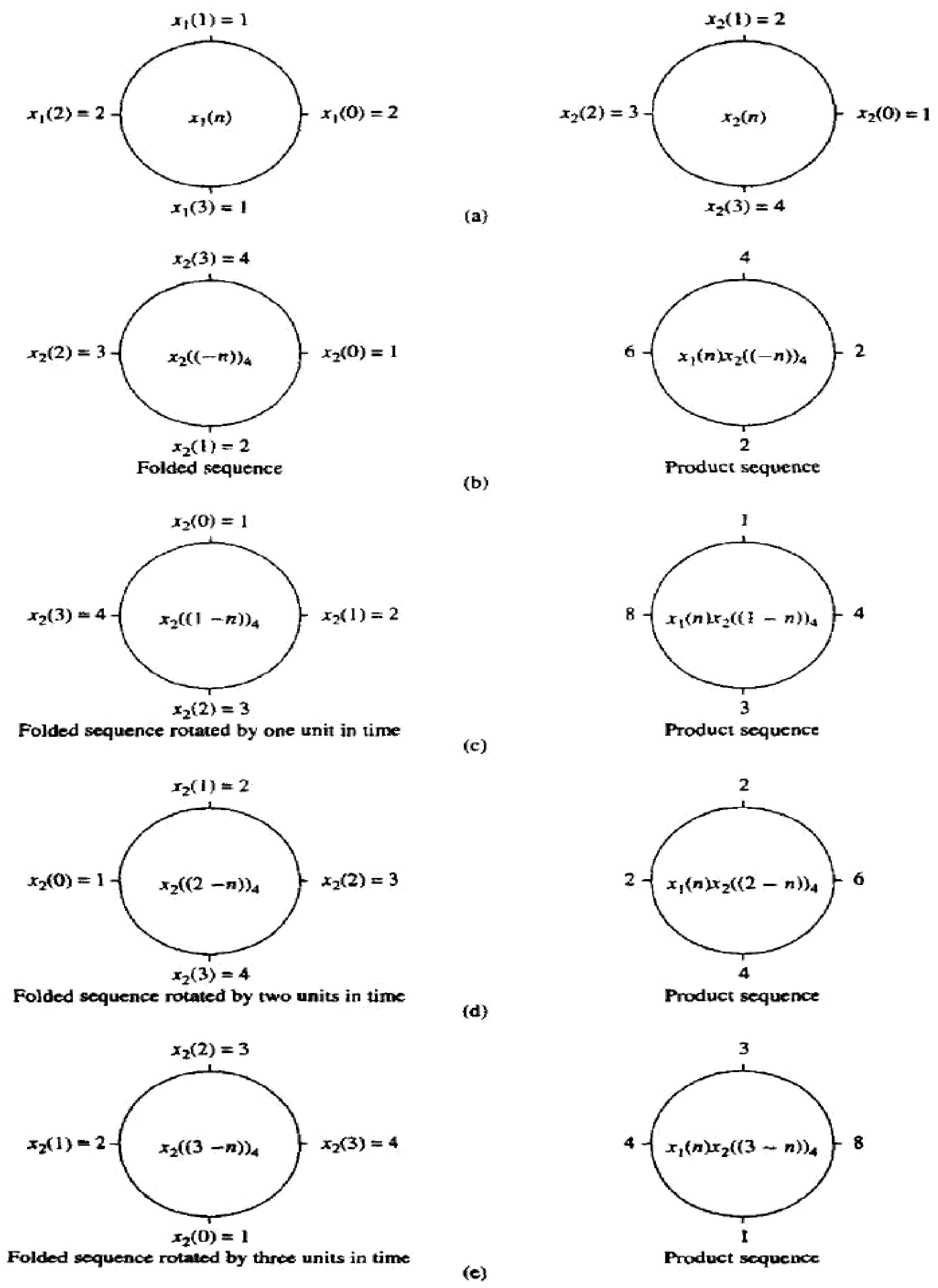


Figure 5.8 Circular convolution of two sequences.

along with the product sequence $x_1(n)x_2((2-n))_4$. By summing the four terms in the product sequence, we obtain

$$x_3(2) = 14$$

For $m = 3$ we have

$$x_3(3) = \sum_{n=0}^{3} x_1(n)x_2((3-n))_4$$

The folded sequence $x_2((-n))_4$ is now rotated by three units in time to yield $x_2((3-n))_4$ and the resultant sequence is multiplied by $x_1(n)$ to yield the product sequence as illustrated in Fig. 5.8(e). The sum of the values in the product sequence is

$$x_3(3) = 16$$

We observe that if the computation above is continued beyond $m = 3$, we simply repeat the sequence of four values obtained above. Therefore, the circular convolution of the two sequences $x_1(n)$ and $x_2(n)$ yields the sequence

$$x_3(n) = \{\underset{\uparrow}{14}, 16, 14, 16\}$$

From this example, we observe that circular convolution involves basically the same four steps as the ordinary *linear convolution* introduced in Chapter 2: *folding* (time reversing) one sequence, *shifting* the folded sequence, *multiplying* the two sequences to obtain a product sequence, and finally, *summing* the values of the product sequence. The basic difference between these two types of convolution is that, in circular convolution, the folding and shifting (rotating) operations are performed in a circular fashion by computing the index of one of the sequences modulo N. In linear convolution, there is no modulo N operation.

The reader can easily show from our previous development that either one of the two sequences may be folded and rotated without changing the result of the circular convolution. Thus

$$x_3(m) = \sum_{n=0}^{N-1} x_2(n)x_1((m-n))_N \qquad m = 0, 1, \ldots, N-1 \tag{5.2.40}$$

The following example serves to illustrate the computation of $x_3(n)$ by means of the DFT and IDFT.

Example 5.2.2

By means of the DFT and IDFT, determine the sequence $x_3(n)$ corresponding to the circular convolution of the sequences $x_1(n)$ and $x_2(n)$ given in Example 5.2.1.

Solution First we compute the DFTs of $x_1(n)$ and $x_2(n)$. The four-point DFT of $x_1(n)$ is

$$\begin{aligned} X_1(k) &= \sum_{n=0}^{3} x_1(n)e^{-j2\pi nk/4} \qquad k = 0, 1, 2, 3 \\ &= 2 + e^{-j\pi k/2} + 2e^{-j\pi k} + e^{-j3\pi k/2} \end{aligned}$$

Thus

$$X_1(0) = 6 \qquad X_1(1) = 0 \qquad X_1(2) = 2 \qquad X_1(3) = 0$$

The DFT of $x_2(n)$ is

$$\begin{aligned} X_2(k) &= \sum_{n=0}^{3} x_2(n) e^{-j2\pi nk/4} \qquad k = 0, 1, 2, 3 \\ &= 1 + 2e^{-j\pi k/2} + 3e^{-j\pi k} + 4e^{-j3\pi k/2} \end{aligned}$$

Thus

$$X_2(0) = 10 \qquad X_2(1) = -2 + j2 \qquad X_2(2) = -2 \qquad X_2(3) = -2 - j2$$

When we multiply the two DFTs, we obtain the product

$$X_3(k) = X_1(k) X_2(k)$$

or, equivalently,

$$X_3(0) = 60 \qquad X_3(1) = 0 \qquad X_3(2) = -4 \qquad X_3(3) = 0$$

Now, the IDFT of $X_3(k)$ is

$$\begin{aligned} x_3(n) &= \sum_{k=0}^{3} X_3(k) e^{j2\pi nk/4} \qquad n = 0, 1, 2, 3 \\ &= \tfrac{1}{4}(60 - 4e^{j\pi n}) \end{aligned}$$

Thus

$$x_3(0) = 14 \qquad x_3(1) = 16 \qquad x_3(2) = 14 \qquad x_3(3) = 16$$

which is the result obtained in Example 5.2.1 from circular convolution.

We conclude this section by formally stating this important property of the DFT.

Circular convolution. If

$$x_1(n) \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} X_1(k)$$

and

$$x_2(n) \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} X_2(k)$$

then

$$x_1(n) \textcircled{N} x_2(n) \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} X_1(k) X_2(k)$$

where $x_1(n) \textcircled{N} x_2(n)$ denotes the circular convolution of the sequence $x_1(n)$ and $x_2(n)$.

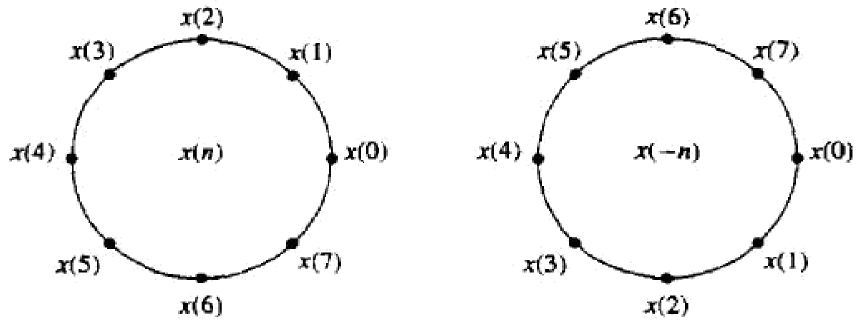


Figure 5.9 Time reversal of a sequence.

Additional DFT Properties

Time reversal of a sequence. If

$$x(n) \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} X(k)$$

then

$$x((-n))_N = x(N-n) \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} X((-k))_N = X(N-k)$$

Hence reversing the N-point sequence in time is equivalent to reversing the DFT values. Time reversal of a sequence $x(n)$ is illustrated in Fig. 5.9.

Proof. From the definition of the DFT in (5.2.1) we have

$$\text{DFT}\{x(N-n)\} = \sum_{n=0}^{N-1} x(N-n)e^{-j2\pi kn/N}$$

If we change the index from n to $m = N - n$, then

$$\begin{aligned}\text{DFT}\{x(N-n)\} &= \sum_{m=0}^{N-1} x(m)e^{-j2\pi k(N-m)/N} \\ &= \sum_{m=0}^{N-1} x(m)e^{j2\pi km/N} \\ &= \sum_{m=0}^{N-1} x(m)e^{-j2\pi m(N-k)/N} = X(N-k)\end{aligned}$$

We note that $X(N-k) = X((-k))_N$, $0 \le k \le N-1$.

Circular time shift of a sequence. If

$$x(n) \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} X(k)$$

then

$$x((n-l))_N \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} X(k)e^{-j2\pi kl/N}$$

Proof. From the definition of the DFT we have

$$\begin{aligned}\text{DFT}\{x((n-l))_N\} &= \sum_{n=0}^{N-1} x((n-l))_N e^{-j2\pi kn/N} \\ &= \sum_{n=0}^{l-1} x((n-l))_N e^{-j2\pi kn/N} \\ &\quad + \sum_{n=l}^{N-1} x(n-l)e^{-j\pi kn/N}\end{aligned}$$

But $x((n-l))_N = x(N-l+n)$. Consequently,

$$\begin{aligned}\sum_{n=0}^{l-1} x((n-l))_N e^{-j2\pi kn/N} &= \sum_{n=0}^{l-1} x(N-l+n)e^{-j2\pi kn/N} \\ &= \sum_{m=N-l}^{N-1} x(m)e^{-j2\pi k(m+l)/N}\end{aligned}$$

Furthermore,

$$\sum_{n=l}^{N-1} x(n-l)e^{-j2\pi kn/N} = \sum_{m=0}^{N-1-l} x(m)e^{-j2\pi k(m+l)/N}$$

Therefore,

$$\begin{aligned}\text{DFT}\{x((n-l))\} &= \sum_{m=0}^{N-1} x(m)e^{-j2\pi k(m+l)/N} \\ &= X(k)e^{-j2\pi kl/N}\end{aligned}$$

Circular frequency shift. If

$$x(n) \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} X(k)$$

then

$$x(n)e^{j2\pi ln/N} \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} X((k-l))_N$$

Hence, the multiplication of the sequence $x(n)$ with the complex exponential sequence $e^{j2\pi kn/N}$ is equivalent to the circular shift of the DFT by l units in frequency. This is the dual to the circular time-shifting property and its proof is similar to the latter.

Complex-conjugate properties. If

$$x(n) \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} X(k)$$

then

$$x^*(n) \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} X^*((-k))_N = X^*(N-k)$$

The proof of this property is left as an exercise for the reader. The IDFT of $X^*(k)$ is

$$\frac{1}{N}\sum_{k=0}^{N-1} X^*(k)e^{j2\pi kn/N} = \left[\frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j2\pi k(N-n)/N}\right]^*$$

Therefore,

$$x^*((-n))_N = x^*(N-n) \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} X^*(k)$$

Circular correlation. In general, for complex-valued sequences $x(n)$ and $y(n)$, if

$$x(n) \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} X(k)$$

and

$$y(n) \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} Y(k)$$

then

$$\tilde{r}_{xy}(l) \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} \tilde{R}_{xy}(k) = X(k)Y^*(k)$$

where $\tilde{r}_{xy}(l)$ is the (unnormalized) circular crosscorrelation sequence, defined as

$$\tilde{r}_{xy}(l) = \sum_{n=0}^{N-1} x(n)y^*((n-l))_N$$

Proof. We can write $\tilde{r}_{xy}(l)$ as the circular convolution of $x(n)$ with $y^*(-n)$, that is,

$$\tilde{r}_{xy}(l) = x(l) \textcircled{N}\, y^*(-l)$$

Then, with the aid of the properties in (5.2.41) and (5.2.46), the N-point DFT of $\tilde{r}_{xy}(l)$ is

$$\tilde{R}_{xy}(k) = X(k)Y^*(k)$$

In the special case where $y(n) = x(n)$, we have the corresponding expression for the circular autocorrelation of $x(n)$,

$$\tilde{r}_{xx}(l) \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} \tilde{R}_{xx}(k) = |X(k)|^2$$

where $\mathbf{W}_N^*$ denotes the complex conjugate of the matrix $\mathbf{W}_N$. Comparison of (5.1.26) with (5.1.25) leads us to conclude that

$$\mathbf{W}_N^{-1} = \frac{1}{N}\mathbf{W}_N^*$$

which, in turn, implies that

$$\mathbf{W}_N\mathbf{W}_N^* = N\mathbf{I}_N$$

where $\mathbf{I}_N$ is an $N \times N$ identity matrix. Therefore, the matrix $\mathbf{W}_N$ in the transformation is an orthogonal (unitary) matrix. Furthermore, its inverse exists and is given as $\mathbf{W}_N^*/N$. Of course, the existence of the inverse of $\mathbf{W}_N$ was established previously from our derivation of the IDFT.

Example 5.1.3

Compute the DFT of the four-point sequence

$$x(n) = (0 \quad 1 \quad 2 \quad 3)$$

Solution The first step is to determine the matrix $\mathbf{W}_4$. By exploiting the periodicity property of $\mathbf{W}_4$ and the symmetry property

$$W_N^{k+N/2} = -W_N^k$$

the matrix $\mathbf{W}_4$ may be expressed as

$$\mathbf{W}_4 = \begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^2 & W_4^4 & W_4^6 \\ W_4^0 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4^1 & W_4^2 & W_4^3 \\ 1 & W_4^2 & W_4^0 & W_4^2 \\ 1 & W_4^3 & W_4^2 & W_4^1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

Then

$$\mathbf{X}_4 = \mathbf{W}_4\mathbf{x}_4 = \begin{bmatrix} 6 \\ -2+2j \\ -2 \\ -2-2j \end{bmatrix}$$

The IDFT of $\mathbf{X}_4$ may be determined by conjugating the elements in $\mathbf{W}_4$ to obtain $\mathbf{W}_4^*$ and then applying the formula (5.1.26).

The DFT and IDFT are computational tools that play a very important role in many digital signal processing applications, such as frequency analysis (spectrum analysis) of signals, power spectrum estimation, and linear filtering. The importance of the DFT and IDFT in such practical applications is due to a large extent on the existence of computationally efficient algorithms, known collectively as fast

6.1.3 Radix-2 FFT Algorithms

In the preceding section we described four algorithms for efficient computation of the DFT based on the divide-and-conquer approach. Such an approach is applicable when the number N of data points is not a prime. In particular, the approach is very efficient when N is highly composite, that is, when N can be factored as $N = r_1 r_2 r_3 \cdots r_\nu$, where the $\{r_j\}$ are prime.

Of particular importance as the case in which $r_1 = r_2 = \cdots = r_\nu \equiv r$, so that $N = r^\nu$. In such a case the DFTs are of size r, so that the computation of the N-point DFT has a regular pattern. The number r is called the radix of the FFT algorithm.

In this section we describe radix-2 algorithms, which are by far the most widely used FFT algorithms. Radix-4 algorithms are described in the following section.

Let us consider the computation of the $N = 2^\nu$ point DFT by the divide-and-conquer approach specified by (6.1.16) through (6.1.18). We select $M = N/2$ and $L = 2$. This selection results in a split of the N-point data sequence into two $N/2$-point data sequences $f_1(n)$ and $f_2(n)$, corresponding to the even-numbered and odd-numbered samples of $x(n)$, respectively, that is,

$$f_1(n) = x(2n)$$

$$f_2(n) = x(2n+1), \qquad n = 0, 1, \ldots, \frac{N}{2} - 1$$

Thus $f_1(n)$ and $f_2(n)$ are obtained by decimating $x(n)$ by a factor of 2, and hence the resulting FFT algorithm is called a decimation-in-time algorithm.

Now the N-point DFT can be expressed in terms of the DFTs of the decimated sequences as follows:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \qquad k = 0, 1, \ldots, N-1$$

$$= \sum_{n \text{ even}} x(n) W_N^{kn} + \sum_{n \text{ odd}} x(n) W_N^{kn}$$

$$= \sum_{m=0}^{(N/2)-1} x(2m) W_N^{2mk} + \sum_{m=0}^{(N/2)-1} x(2m+1) W_N^{k(2m+1)}$$

But $W_N^2 = W_{N/2}$. With this substitution, (6.1.24) can be expressed as

$$X(k) = \sum_{m=0}^{(N/2)-1} f_1(m) W_{N/2}^{km} + W_N^k \sum_{m=0}^{(N/2)-1} f_2(m) W_{N/2}^{km}$$
$$= F_1(k) + W_N^k F_2(k) \qquad k = 0, 1, \ldots, N-1$$

where $F_1(k)$ and $F_2(k)$ are the $N/2$-point DFTs of the sequences $f_1(m)$ and $f_2(m)$, respectively.

Since $F_1(k)$ and $F_2(k)$ are periodic, with period $N/2$, we have $F_1(k + N/2) = F_1(k)$ and $F_2(k + N/2) = F_2(k)$. In addition, the factor $W_N^{k+N/2} = -W_N^k$. Hence (6.1.25) can be expressed as

$$X(k) = F_1(k) + W_N^k F_2(k) \qquad k = 0, 1, \ldots, \frac{N}{2} - 1$$
$$X\left(k + \frac{N}{2}\right) = F_1(k) - W_N^k F_2(k) \qquad k = 0, 1, \ldots, \frac{N}{2} - 1$$

We observe that the direct computation of $F_1(k)$ requires $(N/2)^2$ complex multiplications. The same applies to the computation of $F_2(k)$. Furthermore, there are $N/2$ additional complex multiplications required to compute $W_N^k F_2(k)$. Hence the computation of $X(k)$ requires $2(N/2)^2 + N/2 = N^2/2 + N/2$ complex multiplications. This first step results in a reduction of the number of multiplications from N^2 to $N^2/2 + N/2$, which is about a factor of 2 for N large.

To be consistent with our previous notation, we may define

$$G_1(k) = F_1(k) \qquad k = 0, 1, \ldots, \frac{N}{2} - 1$$
$$G_2(k) = W_N^k F_2(k) \qquad k = 0, 1, \ldots, \frac{N}{2} - 1$$

Then the DFT $X(k)$ may be expressed as

$$X(k) = G_1(k) + G_2(k) \qquad k = 0, 1, \ldots, \frac{N}{2} - 1$$
$$X(k + \frac{N}{2}) = G_1(k) - G_2(k) \qquad k = 0, 1, \ldots, \frac{N}{2} - 1$$

This computation is illustrated in Fig. 6.4.

Having performed the decimation-in-time once, we can repeat the process for each of the sequences $f_1(n)$ and $f_2(n)$. Thus $f_1(n)$ would result in the two

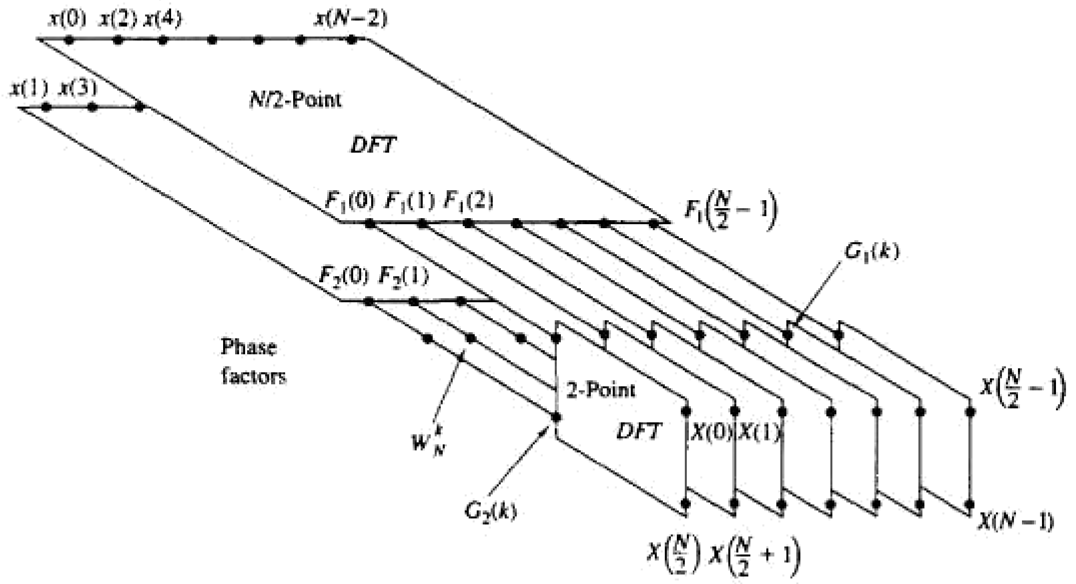


Figure 6.4 First step in the decimation-in-time algorithm.

$N/4$-point sequences

$$v_{11}(n) = f_1(2n) \qquad n = 0, 1, \ldots, \frac{N}{4} - 1$$

$$v_{12}(n) = f_1(2n+1) \qquad n = 0, 1, \ldots, \frac{N}{4} - 1$$

and $f_2(n)$ would yield

$$v_{21}(n) = f_2(2n) \qquad n = 0, 1, \ldots, \frac{N}{4} - 1$$

$$v_{22}(n) = f_2(2n+1) \qquad n = 0, 1, \ldots, \frac{N}{4} - 1$$

By computing $N/4$-point DFTs, we would obtain the $N/2$-point DFTs $F_1(k)$ and $F_2(k)$ from the relations

$$F_1(k) = V_{11}(k) + W_{N/2}^k V_{12}(k) \qquad k = 0, 1, \ldots, \frac{N}{4} - 1$$

$$F_1\left(k + \frac{N}{4}\right) = V_{11}(k) - W_{N/2}^k V_{12}(k) \qquad k = 0, 1, \ldots, \frac{N}{4} - 1$$

$$F_2(k) = V_{21}(k) + W_{N/2}^k V_{22}(k) \qquad k = 0, 1, \ldots, \frac{N}{4} - 1$$

$$F_2\left(k + \frac{N}{4}\right) = V_{21}(k) - W_{N/2}^k V_{22}(k) \qquad k = 0, \ldots, \frac{N}{4} - 1$$

where the $\{V_{ij}(k)\}$ are the $N/4$-point DFTs of the sequences $\{v_{ij}(n)\}$.

TABLE 6.1 COMPARISON OF COMPUTATIONAL COMPLEXITY FOR THE DIRECT COMPUTATION OF THE DFT VERSUS THE FFT ALGORITHM

Number of Points, N	Complex Multiplications in Direct Computation, N^2	Complex Multiplications in FFT Algorithm, $(N/2)\log_2 N$	Speed Improvement Factor
4	16	4	4.0
8	64	12	5.3
16	256	32	8.0
32	1,024	80	12.8
64	4,096	192	21.3
128	16,384	448	36.6
256	65,536	1,024	64.0
512	262,144	2,304	113.8
1,024	1,048,576	5,120	204.8

We observe that the computation of $\{V_{ij}(k)\}$ requires $4(N/4)^2$ multiplications and hence the computation of $F_1(k)$ and $F_2(k)$ can be accomplished with $N^2/4 + N/2$ complex multiplications. An additional $N/2$ complex multiplications are required to compute $X(k)$ from $F_1(k)$ and $F_2(k)$. Consequently, the total number of multiplications is reduced approximately by a factor of 2 again to $N^2/4 + N$.

The decimation of the data sequence can be repeated again and again until the resulting sequences are reduced to one-point sequences. For $N = 2^v$, this decimation can be performed $v = \log_2 N$ times. Thus the total number of complex multiplications is reduced to $(N/2)\log_2 N$. The number of complex additions is $N \log_2 N$. Table 6.1 presents a comparison of the number of complex multiplications in the FFT and in the direct computation of the DFT.

For illustrative purposes, Fig. 6.5 depicts the computation of an $N = 8$ point DFT. We observe that the computation is performed in three stages, beginning with the computations of four two-point DFTs, then two four-point DFTs, and

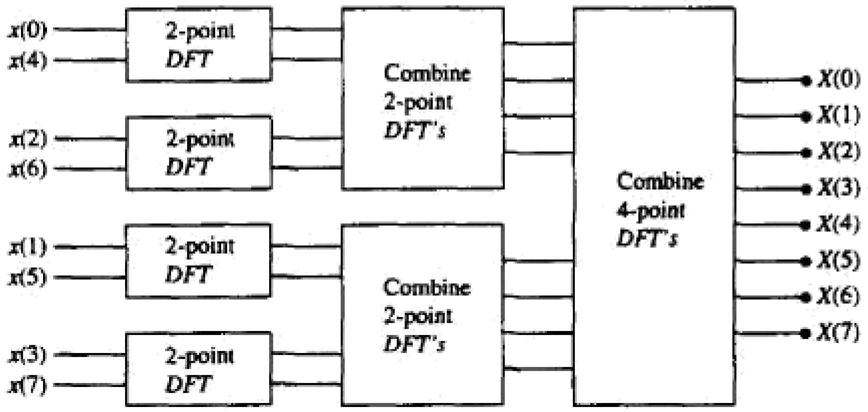


Figure 6.5 Three stages in the computation of an $N = 8$-point DFT.

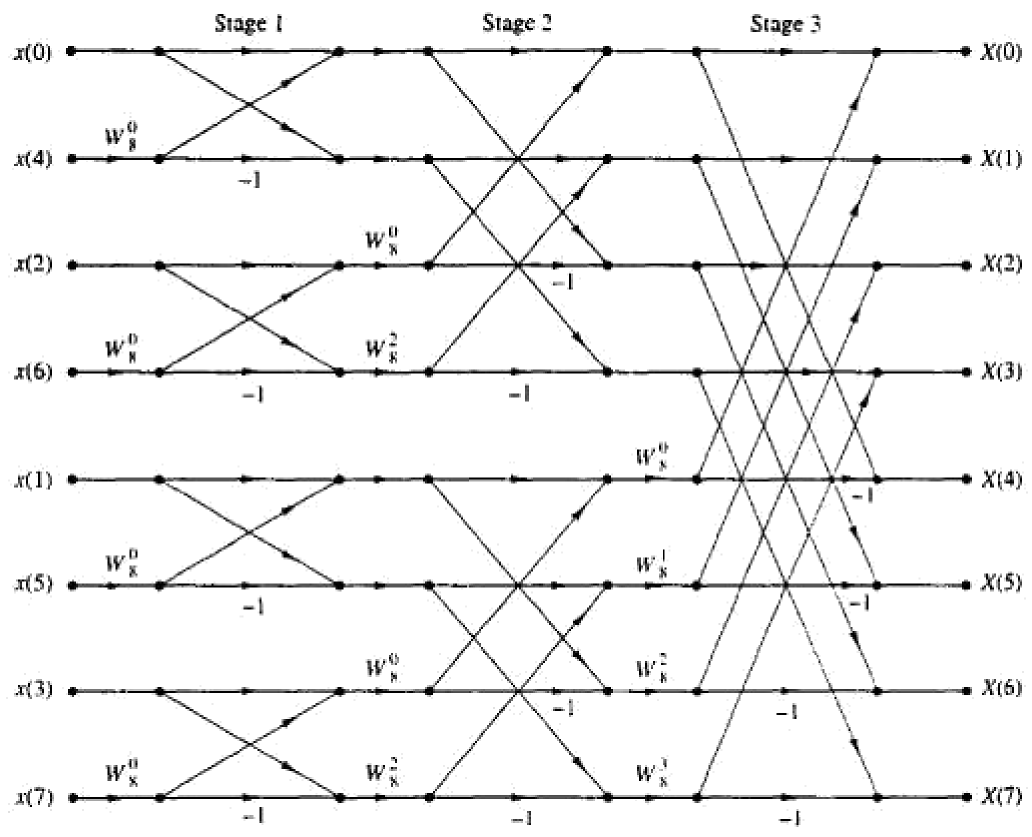


Figure 6.6 Eight-point decimation-in-time FFT algorithm.

finally, one eight-point DFT. The combination of the smaller DFTs to form the larger DFT is illustrated in Fig. 6.6 for $N = 8$.

Observe that the basic computation performed at every stage, as illustrated in Fig. 6.6, is to take two complex numbers, say the pair (a, b), multiply b by W_N^r, and then add and subtract the product from a to form two new complex numbers (A, B). This basic computation, which is shown in Fig. 6.7, is called a *butterfly* because the flow graph resembles a butterfly.

In general, each butterfly involves one complex multiplication and two complex additions. For $N = 2^v$, there are $N/2$ butterflies per stage of the computation process and $\log_2 N$ stages. Therefore, as previously indicated the total number of complex multiplications is $(N/2)\log_2 N$ and complex additions is $N \log_2 N$.

Once a butterfly operation is performed on a pair of complex numbers (a, b) to produce (A, B), there is no need to save the input pair (a, b). Hence we can

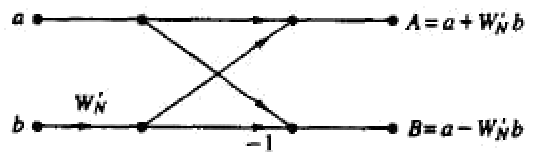


Figure 6.7 Basic butterfly computation in the decimation-in-time FFT algorithm.

store the result (A, B) in the same locations as (a, b). Consequently, we require a fixed amount of storage, namely, $2N$ storage registers, in order to store the results (N complex numbers) of the computations at each stage. Since the same $2N$ storage locations are used throughout the computation of the N-point DFT, we say that *the computations are done in place.*

A second important observation is concerned with the order of the input data sequence after it is decimated $(\nu - 1)$ times. For example, if we consider the case where $N = 8$, we know that the first decimation yields the sequence $x(0)$, $x(2)$, $x(4)$, $x(6)$, $x(1)$, $x(3)$, $x(5)$, $x(7)$, and the second decimation results in the sequence $x(0)$, $x(4)$, $x(2)$, $x(6)$, $x(1)$, $x(5)$, $x(3)$, $x(7)$. This *shuffling* of the input data sequence has a well-defined order as can be ascertained from observing Fig. 6.8, which illustrates the decimation of the eight-point sequence. By expressing the index n, in the sequence $x(n)$, in binary form, we note that the order of the decimated data sequence is easily obtained by reading the binary representation of the index n in reverse order. Thus the data point $x(3) \equiv x(011)$ is placed in position $m = 110$ or $m = 6$ in the decimated array. Thus we say that the data $x(n)$ after decimation is stored in bit-reversed order.

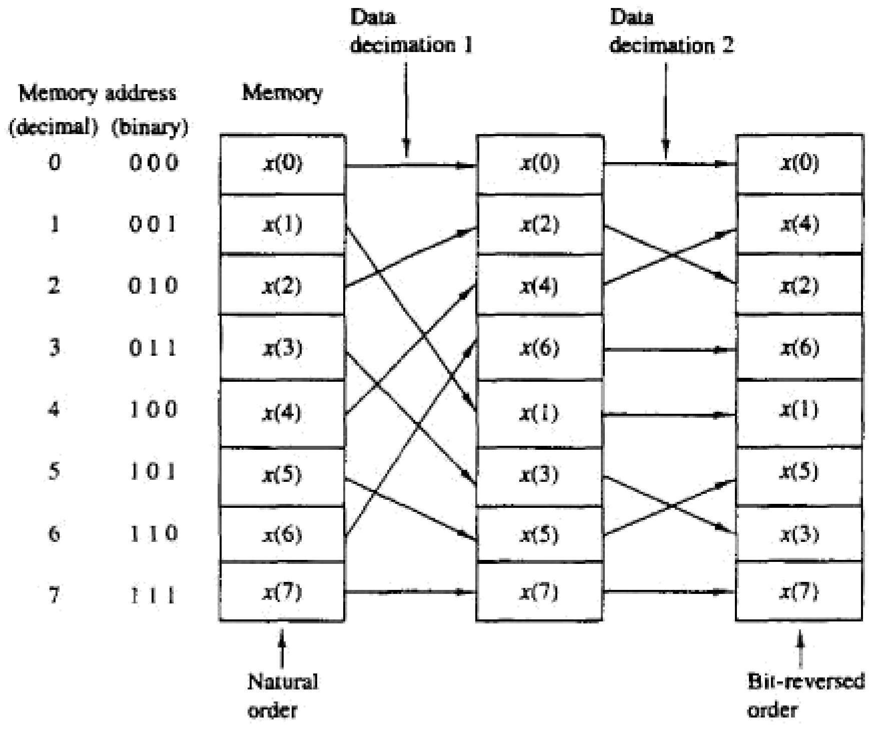
With the input data sequence stored in bit-reversed order and the butterfly computations performed in place, the resulting DFT sequence $X(k)$ is obtained in natural order (i.e., $k = 0, 1, \ldots, N - 1$). On the other hand, we should indicate that it is possible to arrange the FFT algorithm such that the input is left in natural order and the resulting output DFT will occur in bit-reversed order. Furthermore, we can impose the restriction that both the input data $x(n)$ and the output DFT $X(k)$ be in natural order, and derive an FFT algorithm in which the computations are not done in place. Hence such an algorithm requires additional storage.

Another important radix-2 FFT algorithm, called the decimation-in-frequency algorithm, is obtained by using the divide-and-conquer approach described in Section 6.1.2 with the choice of $M = 2$ and $L = N/2$. This choice of parameters implies a column-wise storage of the input data sequence. To derive the algorithm, we begin by splitting the DFT formula into two summations, one of which involves the sum over the first $N/2$ data points and the second sum involves the last $N/2$ data points. Thus we obtain

$$\begin{aligned} X(k) &= \sum_{n=0}^{(N/2)-1} x(n)W_N^{kn} + \sum_{n=N/2}^{N-1} x(n)W_N^{kn} \\ &= \sum_{n=0}^{(N/2)-1} x(n)W_N^{kn} + W_N^{Nk/2} \sum_{n=0}^{(N/2)-1} x\left(n + \frac{N}{2}\right) W_N^{kn} \end{aligned}$$

Since $W_N^{kN/2} = (-1)^k$, the expression (6.1.33) can be rewritten as

$$X(k) = \sum_{n=0}^{(N/2)-1} \left[x(n) + (-1)^k x\left(n + \frac{N}{2}\right)\right] W_N^{kn}$$



(a)

$(n_2n_1n_0)$	$\rightarrow$	$(n_0n_2n_1)$	$\rightarrow$	$(n_0n_1n_2)$
(0 0 0)	$\rightarrow$	(0 0 0)	$\rightarrow$	(0 0 0)
(0 0 1)	$\rightarrow$	(1 0 0)	$\rightarrow$	(1 0 0)
(0 1 0)	$\rightarrow$	(0 0 1)	$\rightarrow$	(0 1 0)
(0 1 1)	$\rightarrow$	(1 0 1)	$\rightarrow$	(1 1 0)
(1 0 0)	$\rightarrow$	(0 1 0)	$\rightarrow$	(0 0 1)
(1 0 1)	$\rightarrow$	(1 1 0)	$\rightarrow$	(1 0 1)
(1 1 0)	$\rightarrow$	(0 1 1)	$\rightarrow$	(0 1 1)
(1 1 1)	$\rightarrow$	(1 1 1)	$\rightarrow$	(1 1 1)

(b)

Figure 6.8 Shuffling of the data and bit reversal.

Now, let us split (decimate) $X(k)$ into the even- and odd-numbered samples. Thus we obtain

$$X(2k) = \sum_{n=0}^{(N/2)-1} \left[x(n) + x\left(n + \frac{N}{2}\right)\right] W_{N/2}^{kn} \qquad k = 0, 1, \ldots, \frac{N}{2} - 1$$

and

$$X(2k+1) = \sum_{n=0}^{(N/2)-1} \left\{\left[x(n) - x\left(n + \frac{N}{2}\right)\right] W_N^n\right\} W_{N/2}^{kn} \qquad k = 0, 1, \ldots, \frac{N}{2} - 1$$

where we have used the fact that $W_N^2 = W_{N/2}$.

If we define the $N/2$-point sequences $g_1(n)$ and $g_2(n)$ as

$$g_1(n) = x(n) + x\left(n + \frac{N}{2}\right)$$

$$g_2(n) = \left[x(n) - x\left(n + \frac{N}{2}\right)\right] W_N^n \qquad n = 0, 1, 2, \ldots, \frac{N}{2} - 1$$

then

$$X(2k) = \sum_{n=0}^{(N/2)-1} g_1(n) W_{N/2}^{kn}$$

$$X(2k+1) = \sum_{n=0}^{(N/2)-1} g_2(n) W_{N/2}^{kn}$$

The computation of the sequences $g_1(n)$ and $g_2(n)$ according to (6.1.37) and the subsequent use of these sequences to compute the $N/2$-point DFTs are depicted in Fig. 6.9. We observe that the basic computation in this figure involves the butterfly operation illustrated in Fig. 6.10.

This computational procedure can be repeated through decimation of the $N/2$-point DFTs, $X(2k)$ and $X(2k+1)$. The entire process involves $\nu = \log_2 N$

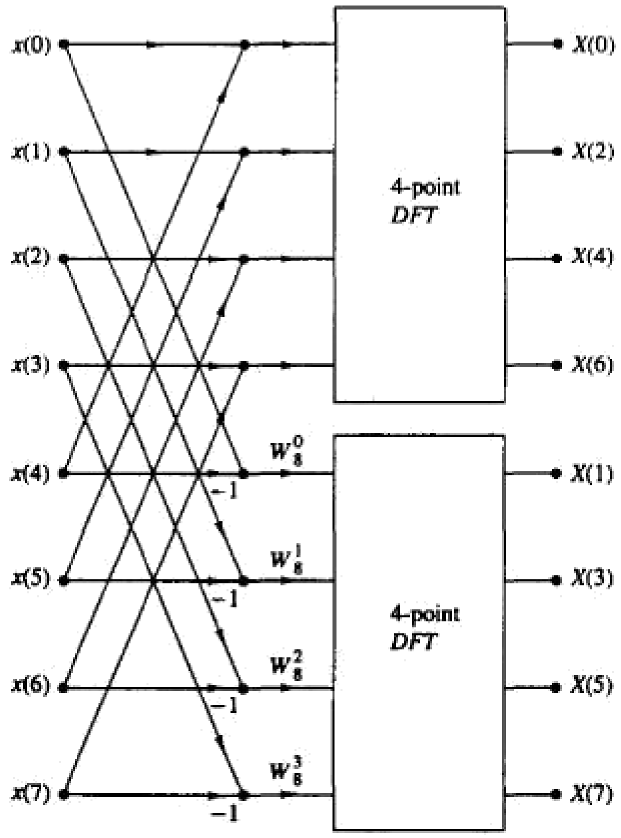


Figure 6.9 First stage of the decimation-in-frequency FFT algorithm.

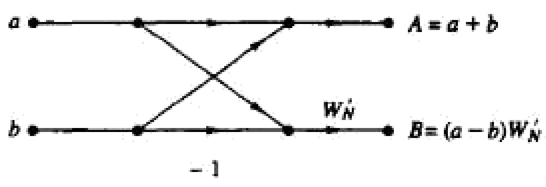


Figure 6.10 Basic butterfly computation in the decimation-in-frequency FFT algorithm.

stages of decimation, where each stage involves $N/2$ butterflies of the type shown in Fig. 6.10. Consequently, the computation of the N-point DFT via the decimation-in-frequency FFT algorithm, requires $(N/2)\log_2 N$ complex multiplications and $N\log_2 N$ complex additions, just as in the decimation-in-time algorithm. For illustrative purposes, the eight-point decimation-in-frequency algorithm is given in Fig. 6.11.

We observe from Fig. 6.11, that the input data $x(n)$ occurs in natural order, but the output DFT occurs in bit-reversed order. We also note that the computations are performed in place. However, it is possible to reconfigure the decimation-in-frequency algorithm so that the input sequence occurs in bit-reversed order while the output DFT occurs in normal order. Furthermore, if we abandon the requirement that the computations be done in place, it is also possible to have both the input data and the output DFT in normal order.

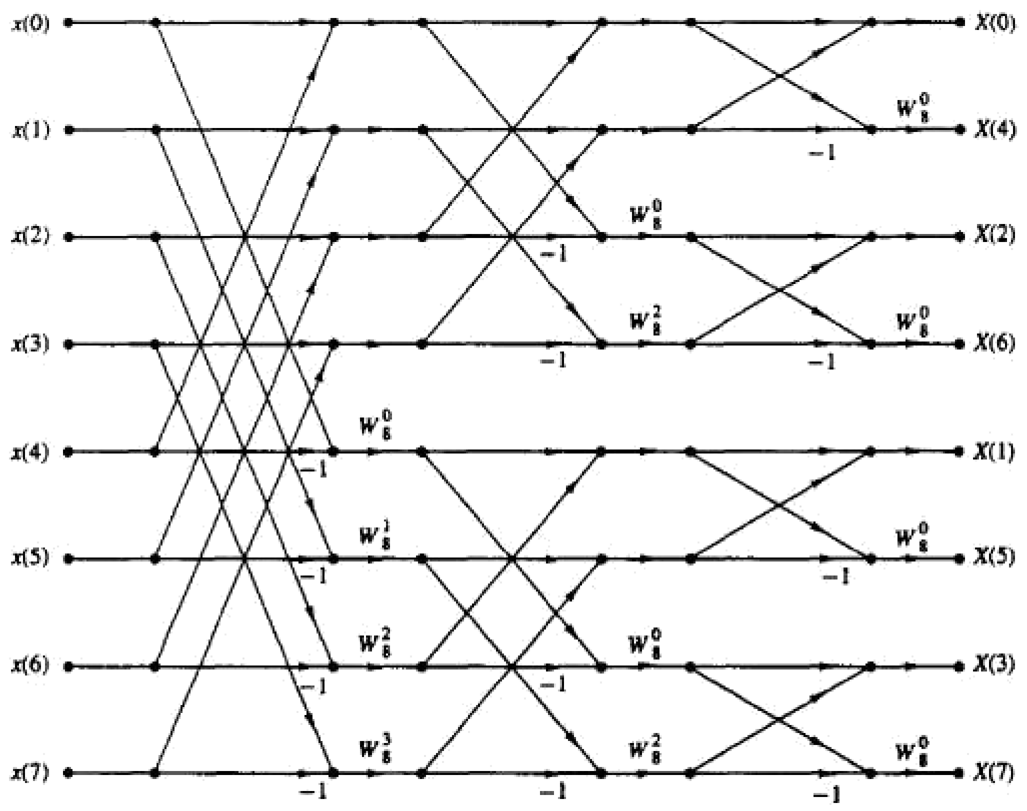


Figure 6.11 N = 8-point decimation-in-frequency FFT algorithmn.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL AND ELECTRONICS

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

UNIT – II - – DESIGN OF INFINITE IMPULSE RESPONSE FILTER-SEC1315

2.1 INTRODUCTION

To remove or to reduce strength of unwanted signal like noise and to improve the quality of required signal filtering process is used. To use the channel full bandwidth we mix up two or more signals on transmission side and on receiver side we would like to separate it out in efficient way. Hence filters are used. Thus the digital filters are mostly used in

1. Removal of undesirable noise from the desired signals
2. Equalization of communication channels
3. Signal detection in radar, sonar and communication
4. Performing spectral analysis of signals.

Analog and digital filters

In signal processing, the function of a filter is to remove unwanted parts of the signal, such as random noise, or to extract useful parts of the signal, such as the components lying within a certain frequency range. The following block diagram illustrates the basic idea.

There are two main kinds of filter, analog and digital. They are quite different in their physical makeup and in how they work. An analog filter uses analog electronic circuits made up from components such as resistors, capacitors and op amps to produce the required filtering effect. Such filter circuits are widely used in such applications as noise reduction, video signal enhancement, graphic equalizers in hi-fi systems, and many other areas.

In analog filters the signal being filtered is an electrical voltage or current which is the direct analogue of the physical quantity (e.g. a sound or video signal or transducer output) involved.

A digital filter uses a digital processor to perform numerical calculations on sampled values of the signal.

The processor may be a general-purpose computer such as a PC, or a specialized DSP (Digital Signal Processor) chip. The analog input signal must first be sampled and digitized using an ADC (analog to digital converter). The resulting binary numbers, representing successive sampled values of the input signal, are transferred to the processor, which carries out numerical calculations on them. These calculations typically involve multiplying the input values by constants and adding the products together. If necessary, the results of these calculations, which now represent sampled values of the filtered signal, are output through a DAC (digital to analog converter) to convert the signal back to analog form. In a digital filter, the signal is represented by a sequence of numbers, rather than a voltage or current.

.1 INTRODUCTION

The filters designed by considering all the infinite samples of impulse response are called IIR filters. The impulse response is obtained by taking inverse fourier transform of ideal frequency response. The popular methods for such filter design uses the technique of transforming the analog filter to an equivalent digital filter. We know that the analog filter with transfer function $H_a(s)$ is stable if all its poles lie in the left half of the s-plane. Consequently, if the conversion technique is to be effective, it should posses the following desirable properties.

1. The imaginary axis in the s-plane should map into the unit circle in the z-plane. Thus there will be a direct relationship between the two frequency variables in the two domains.
2. The left-half of the s-plane should map into the interior of the unit circle in the z-plane. Thus a stable analog filter will be converted to a stable digital filter.

The IIR filter is a discrete time system that is designed to pass the spectral content of the input signal in a specified band of frequencies. Based on the frequency response the filters are classified into four types. They are lowpass, highpass, bandpass and bandstop filters. The ideal frequency response of the filters are shown by solid lines in fig 4.1 Since the ideal response is not realizable they are approximated using a filter approximation function. The approximation problem is solved to meet a specified tolerance in the passband and stopband. The shaded areas in the fig 4.1 shows the tolerance regions of the ideal frequency response. In the passband the

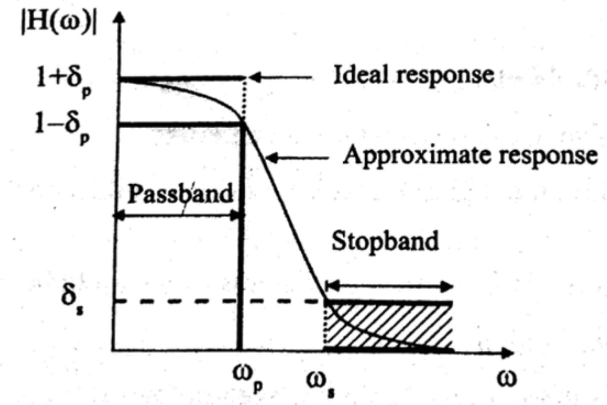


Fig a:Lowpass filter

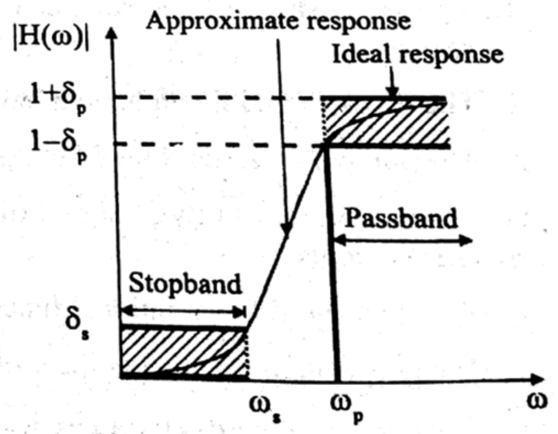


Fig b : Highpass filter

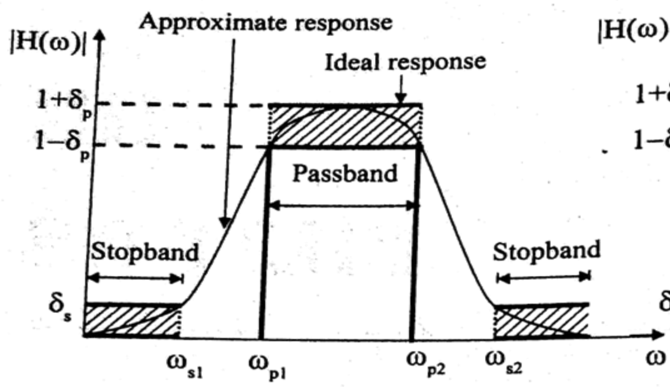


Fig c : Bandpass filter

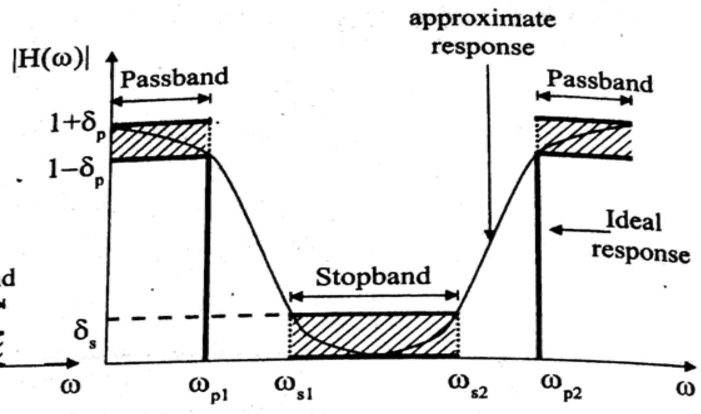


Fig d : Bandstop filter

Fig .1 : Ideal and approximate magnitude response of filters.

magnitude is approximated to unity within an error of $\pm\delta_p$. In the stopband the magnitude is approximated to zero within an error of δ_s. Here the δ_p and δ_s are the limits of the tolerance in the passband and stopband. The δ_p and δ_s are also called ripples. The frequencies ω_p and ω_s represent the passband and stopband edge frequencies respectively.

A number of solutions to the approximation problem of analog filter design are well developed. The popular among them are butterworth and Chebyshev approximation. The approximation problem for digital filter design is conceptually same as that of analog filter design. Hence for designing a digital IIR filter, first an equivalent analog filter is designed using any one of the approximation technique and the given specifications. The result of the analog filter design will be an analog filter transfer function $H_a(s)$. The analog transfer function is converted to digital transfer function H(z) using either Bilinear or Impulse invariant transformation. The digital transfer function H(z) can be realized in a software that runs on a digital hardware (or it can be implemented in firmware).

The designed transfer function of the filter should represent a stable and causal system. For stability and causality of analog filter, the analog transfer function should satisfy the following requirements.

1. The $H_a(s)$ should be a rational function of s and the coefficients of s should be real.
2. The poles should lie on the left half of s-plane.
3. The number of zeros should be less than or equal to number of poles.

For stability and causality of digital filter, the digital transfer function should satisfy the following requirements.

1. The H(z) should be a rational function of z and the coefficients of z should be real.
2. The poles should lie inside the unit circle in z-plane.
3. The number of zeros should be less than or equal to number of poles.

Table .1 : Comparison of digital and analog filters.

Digital Filter	Analog Filter
1. Operates on digital samples (or sampled version) of the signal.	1. Operates on analog signals (or actual signals).
2. It is governed (or defined) by linear difference equation.	2. It is governed (or defined) by linear differential equation.
3. It consists of adders, multipliers and delays implemented in digital logic (either in hardware or software or both)	3. It consists of electrical components like resistors, capacitors and inductors.
4. In digital filters the filter coefficients are designed to satisfy the desired frequency response.	4. In analog filters the approximation problem is solved to satisfy the desired frequency response.

Advantages of digital filters

1. The values of resistors,capacitors and inductors used in the analog filters changes with temperature. Since the digital filters does not have these components, they have high thermal stability.
2. In digital filters the precision of the filter depends on the length (or size) of the registers used to store the filter coefficients. Hence by increasing the register bit - length (in hardware) the performance characteristics of the filter like accuracy, dynamic range, stability and frequency response tolerance, can be enhanced.
3. The digital filters are programmable. Hence the filter coefficients can be changed any time to implement adaptive features.
4. A single filter can be used to process multiple signals by using the techniques of multiplexing.

Disadvantages of digital filters

1. The bandwidth of the discrete signal is limited by the sampling frequency. The bandwidth of real discrete signal is half the sampling frequency.
2. The performance of the digital filter depends on the hardware (i.e., depends on the bit length of the registers in the hardware) used to implement the filter.

Important features of IIR filters

1. The physically realizable IIR filters does not have linear phase.
2. The IIR filter specifications includes the desired characteristics for the magnitude response only.

Let $\alpha_1 = \frac{1}{A_1} = \frac{1}{|H(\omega)|_{\omega=\omega_1}}$ and $\alpha_2 = \frac{1}{A_2} = \frac{1}{|H(\omega)|_{\omega=\omega_2}}$

Here α_1 is called attenuation at the passband frequency ω_1 and α_2 is called attenuation at the stopband frequency ω_2.

The maximum value of normalized gain is unity and so, A_1 & A_2 are less than 1 and α_1 & α_2 are greater than 1. The magnitude response of lowpass filter in terms of gain and attenuation are shown in fig 4.8. In this sketch A_1 is assumed as 0.8 and A_2 is assumed as 0.2 Hence $\alpha_1 = 1/0.8 = 1.25$ and $\alpha_2 = 1/0.2 = 5$.

Another popular unit that is used for filter specification is dB. When the gain is expressed in dB it will be a negative dB. When the attenuation is expressed in dB it will be a positive dB.

Let k_1 = Gain in dB at a Passband frequency ω_1.
k_2 = Gain in dB at a Stopband frequency ω_2.

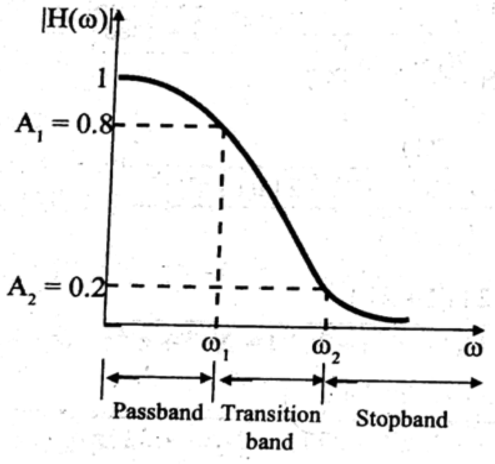


Fig a : Gain Vs ω

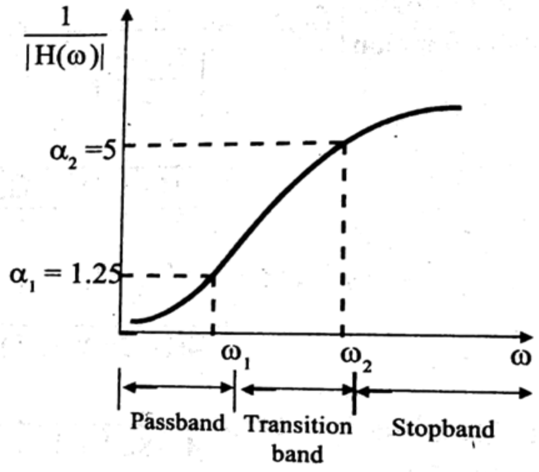


Fig b : Attenuation Vs ω

Fig 4.8 : Magnitude response of Low pass filter

The gain can be converted to normal values as shown below.

$20 \log A_1 = k_1$	$20 \log A_2 = k_2$
$\log A_1 = k_1/20$	$\log A_2 = k_2/20$
$\therefore A_1 = 10^{k_1/20}$	$\therefore A_2 = 10^{k_2/20}$

When expressed in dB the gain and attenuation will have only change in sign, because $\log \alpha = \log(1/A) = -\log A$. (Hence when the dB is positive it is attenuation and when the dB is negative it is gain).

When $A_1 = 0.8$, $k_1 = 20 \log A_1 = 20 \log 0.8 = -1.9382 \approx -2\text{dB}$
When $A_2 = 0.2$, $k_2 = 20 \log A_2 = 20 \log 0.2 = -13.9794 \approx -14\text{ dB}$

The magnitude response of lowpass filter interms of db-gain and db-magnitude are shown in fig 4.9.

SPECIFICATIONS OF THE LOW PASS FILTER

Let w_1=pass band digital frequency in rad/sec ,w_2=stop band digital frequency in rad/sec , A_1=gain in pass band ,A_2=gain in stop band

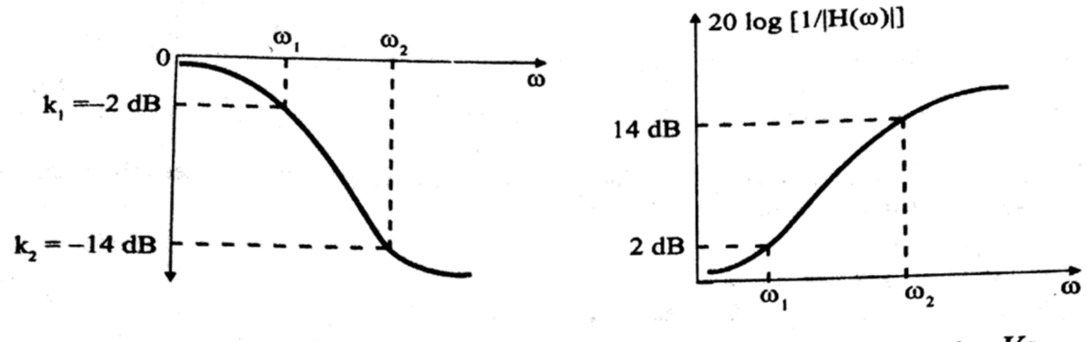


***Fig a** : dB-Gain Vs ω* ***Fig b** : dB - Attenuation Vs ω*

***Fig 4.9** : Magnitude response of lowpass filter*

Sometimes the specifications are given in terms of passband ripple δ_p and stopband ripple δ_s. In this case the dB gain and attenuation can be estimated as shown below.

$$k_1 = 20 \log (1-\delta_p) \qquad \alpha_1 = -20 \log (1-\delta_p)$$
$$k_2 = 20 \log \delta_s \qquad \alpha_2 = -20 \log \delta_s$$

***Note** : If the ripples are specified in dB, then the minimum passband ripple is equal to k_1 and the negative of maximum passband attenuation is equal to k_2.*

DESIGN OF LOWPASS DIGITAL BUTTERWORTH FILTER

The popular methods of designing IIR digital filter involves the design of equivalent analog filter and then converting the analog filter to digital filter. Hence to design a butterworth IIR digital filter, first an analog butterworth filter transfer function is determined using the given specifications. Then the analog filter transfer function is converted to a digital filter transfer function by using either impulse invariance transformation or bilinear transformation.

Analog Butterworth filter

The analog butterworth filter is designed by approximating the ideal frequency response using an error function. The error function is selected such that the magnitude is maximally flat in the passband and monotonically decreasing in the stopband. (Stictly saying, the magnitude is maximally flat at the origin i.e., at $\Omega = 0$, and monotonically decreasing with increasing Ω)

The magnitude response of lowpass filter obtained by this approximation is given by

$$|H_a(\Omega)|^2 = \frac{1}{1+\left(\frac{\Omega}{\Omega_c}\right)^{2N}}$$

We know that the frequency response $H_a(\Omega)$ of an analog filter is obtained by letting $s = j\Omega$ in the analog transfer function $H_a(\Omega)$. Hence substituting Ω by s/j in equation gives the system transfer function.

$$\therefore H_a(s)H_a(-s) = \frac{1}{1+\left(\frac{s/j}{\Omega_c}\right)^{2N}} = \frac{1}{1+\left(\frac{s^2}{j^2\,\Omega_c^2}\right)^{N}}$$

In equation (4.36), when s/Ω_c is replaced by s_n (i.e., letting $\Omega_c = 1$ rad/sec) the transfer function is called normalized transfer function.

$$\therefore H_a(s_n)\,H_a(-s_n) = \frac{1}{1+\left(-s_n^2\right)^N}$$

The transfer function of equation (4.37) will have 2N poles which are given by the roots of the denominator polynomial. It can be shown that the poles of the transfer function symmetrically lies on a unit circle in s-plane with angular spacing of π/N.

For a stable and causal filter the poles should lie on the left half of s-plane. Hence the desired filter transfer function is formed by choosing the N-number of left half poles. When N is even, all the poles are complex and exist as conjugate pair. When N is odd, one of the pole is real and all other poles are complex and exist as conjugate pair. Therefore the transfer function of butterworth filters will be a product of second order factors. The analog filter transfer function of normalized and unnormalized butterworth lowpass filters are given below.

Normalized butterworth lowpass filter transfer function

Let N be the order of the filter.

When N is even, $H_a(s) = \prod_{k=1}^{\frac{N}{2}} \frac{1}{s_n^2 + b_k s_n + 1}$

When N is odd, $H_a(s) = \frac{1}{s_n+1}\prod_{k=1}^{\frac{N-1}{2}} \frac{1}{s_n^2 + b_k s_n + 1}$

$$\text{where, } b_k = 2\sin\left[\frac{(2k-1)\pi}{2N}\right]$$

Unnormalized butterworth lowpass filter transfer function

The unnormalized transfer function is obtained by replacing s_n by s/Ω_c, where Ω_c is the 3-dB cutoff frequency of the lowpass filter.

Let N be the order of the filter.

When N is even, $H_a(s) = \prod_{k=1}^{\frac{N}{2}} \frac{\Omega_c^2}{s^2 + b_k\Omega_c s + \Omega_c^2}$

$$\text{When N is odd, } H_a(s) = \frac{\Omega_c}{s+\Omega_c} \prod_{k=1}^{\frac{N-1}{2}} \frac{\Omega_c^2}{s^2 + b_k\Omega_c + \Omega_c^2}$$

$$\text{where, } b_k = 2\sin\left[\frac{(2k-1)\pi}{2N}\right]$$

Frequency response of butterworth filter

The frequency response of butterworth filter depends on the order N. The magnitude response (frequency response) for different values of N are shown in fig 4.10. From fig 4.10 it can be observed that the approximated magnitude response approaches the ideal response as the value of N increases.

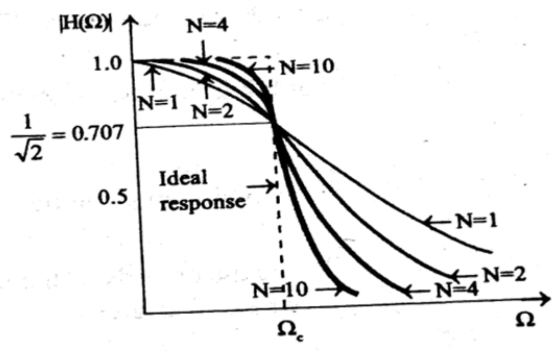


Fig. 4.10: Magnitude response of butterworth lowpass filter for various values of N

Order of the filter

In butterworth filters the frequency response of the filter depends on the order, N. Hence the order N has to be estimated to satisfy the given specifications.

Usually the specifications of the filter are given in terms of gain or attenuation at a passband and stopband frequency.

Let, A_1 = Gain or Magnitude at a passband frequency Ω_1.

A_2 = Gain or Magnitude at a stopband frequency Ω_2.

Calculate a parameter N_1 using equation (4.44) and correct it to nearest integer. Choose N such that $N \geq N_1$.

$$N_1 = \frac{1}{2}\,\frac{\log\left\{\left[\frac{1}{A_1^2}-1\right]\Big/\left[\frac{1}{A_2^2}-1\right]\right\}}{\log\left(\frac{\Omega_2}{\Omega_1}\right)}$$

Properties of butterworth filters

1. The Butterworth filters are all pole designs. (i.e., the zeros of the filters exist at infinity).
2. At the cutoff frequency Ω_c the magnitude of normalized butterworth filter is $1/\sqrt{2}$ (i.e., $|H_a(\Omega)| = 1/\sqrt{2} = 0.707$). Hence the dB magnitude at the cutoff frequency wil be 3 dB less than the maximum value.
3. The filter order N completely specifies the filter.
4. The magnitude is maximally flat at the origin.

When N is odd, $$H_a(s) = \frac{\Omega_c}{s+\Omega_c}\prod_{k=1}^{\frac{N-1}{2}} \frac{\Omega_c^2}{s^2+b_k\Omega_c+\Omega_c^2}$$

$$\text{where, } b_k = 2\sin\left[\frac{(2k-1)\pi}{2N}\right]$$

Frequency response of butterworth filter

The frequency response of butterworth filter depends on the order N. The magnitude response (frequency response) for different values of N are shown in fig 4.10. From fig 4.10 it can be observed that the approximated magnitude response approaches the ideal response as the value of N increases.

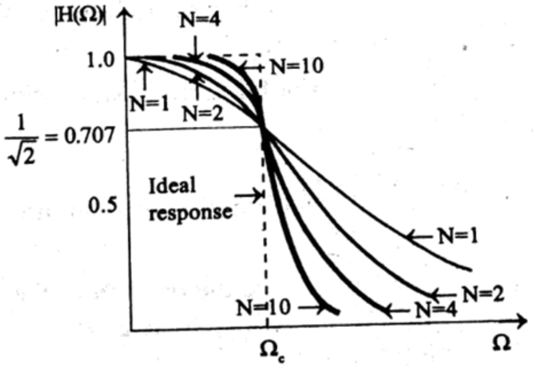


Fig. 4.10: Magnitude response of butterworth lowpass filter for various values of N

Order of the filter

In butterworth filters the frequency response of the filter depends on the order, N. Hence the order N has to be estimated to satisfy the given specifications.

Usually the specifications of the filter are given in terms of gain or attenuation at a passband and stopband frequency.

Let, A_1 = Gain or Magnitude at a passband frequency Ω_1.

A_2 = Gain or Magnitude at a stopband frequency Ω_2.

Calculate a parameter N_1 using equation (4.44) and correct it to nearest integer. Choose N such that $N \geq N_1$.

$$N_1 = \frac{1}{2}\,\frac{\log\left\{\left[\frac{1}{A_1^2}-1\right]\Big/\left[\frac{1}{A_2^2}-1\right]\right\}}{\log\left(\frac{\Omega_2}{\Omega_1}\right)}$$

Properties of butterworth filters

1. The Butterworth filters are all pole designs. (i.e., the zeros of the filters exist at infinity).
2. At the cutoff frequency Ω_c the magnitude of normalized butterworth filter is $1/\sqrt{2}$ (i.e., $|H_a(\Omega)| = 1/\sqrt{2} = 0.707$). Hence the dB magnitude at the cutoff frequency will be 3 dB less than the maximum value.
3. The filter order N completely specifies the filter.
4. The magnitude is maximally flat at the origin.

5. The magnitude is a monotonically decreasing function of Ω.

6. The magnitude response approaches the ideal response as the value of N increases.

Design procedure for lowpass digital butterworth IIR filter

A_1 = Gain at a passband frequency ω_1.
A_2 = Gain at a stopband frequency ω_2.
Ω_1 = Analog frequency corresponding to ω_1.
Ω_2 = Analog frequency corresponding to ω_2.

1. Choose either bilinear or impulse invariant transformation.

2. Calculate the ratio of Ω_2/Ω_1.

For bilinear transformation, $\dfrac{\Omega_2}{\Omega_1} = \dfrac{\tan \omega_2 / 2}{\tan \omega_1 / 2}$

For impulse invariant transformation, $\dfrac{\Omega_2}{\Omega_1} = \dfrac{\omega_2}{\omega_1}$

3. Decide the order N of the filter. The order N should be greater than or equal to N_1, where N_1 is given by

$$N_1 = \frac{1}{2}\,\frac{\log\left\{\left[\frac{1}{A_2^2}-1\right]\Big/\left[\frac{1}{A_1^2}-1\right]\right\}}{\log\frac{\Omega_2}{\Omega_1}}$$

(Choose N such that, $N \geq N_1$)

4. Calculate the analog cutoff frequency Ω_c.

For bilinear transformation, $\Omega_c = \dfrac{2}{T}\,\dfrac{\tan \omega_1 / 2}{\left[\dfrac{1}{A_1^2}-1\right]^{\frac{1}{2N}}}$

For impulse invariant transformation, $\Omega_c = \dfrac{\omega_1 / T}{\left[\dfrac{1}{A_1^2}-1\right]^{\frac{1}{2N}}}$

5. Determine the analog transfer function of the filter.

Let, $H_a(s)$ = Analog filter transfer function.

When the order N is even for unity dc gain filter, $H_a(s)$ is given by equation

$$H_a(s) = \prod_{k=1}^{\frac{N}{2}} \frac{\Omega_c^2}{s^2 + b_k\Omega_c s + \Omega_c^2}$$

When the order N is odd for unity dc gain filter, $H_a(s)$ is given by equation

$$H_a(s) = \frac{\Omega_c}{s+\Omega_c} \prod_{k=1}^{\frac{N-1}{2}} \frac{\Omega_c^2}{s^2 + b_k\Omega_c s + \Omega_c^2}$$

The coefficient b_k is given by,

$$b_k = 2\sin\left(\frac{(2k-1)\pi}{2N}\right)$$

Note : *For normalized case take Ω_c = 1 rad/sec.*

6. Using the chosen transformation, transform $H_a(s)$ to H(z), where H(z) is the transfer function of the digital filter.
7. Realize the digital filter transfer function H(z) by a suitable structure.

Note : *The basic filter design is lowpass filter design, The highpass, bandpass or bandstop filters are obtained from lowpass filter design by frequency transformation.*

DESIGN OF LOWPASS DIGITAL CHEBYSHEV FILTER

For designing a chebyshev IIR digital filter, first an analog filter is designed using the given specifications. Then the analog filter transfer function is transformed to digital filter transfer function by using either impulse invariance transformation or bilinear transformation.

Analog Chebyshev filter

The analog Chebyshev filter is designed by approximating the ideal frequency response using an error function. The approximation function is selected such that the error is minimized over a prescribed band of frequencies. There are two types of Chebyshev approximation. In type-1 approximation, the error function is selected such that, the magnitude response is equiripple in the passband and monotonic in the stopband. In type-2 approximation the error function is selected such that, the magnitude response is monotonic in passband and equiripple in stopband. The type-2 magnitude response is also called inverse Chebyshev response. The type-1 design is presented in this book.

The magnitude response of Type-1 lowpass filter is given by

$$|H_a(\Omega)|^2 = \frac{1}{1+ \epsilon^2\, C_N^2\left(\frac{\Omega}{\Omega_c}\right)}$$

where $\in$ is attenuation constant and $C_N(\Omega/\Omega_c)$ is the Chebyshev polynomial of the first kind of degree N.

The attenuation constant, $\in = \left[\frac{1}{A_1^2} - 1\right]^{\frac{1}{2}}$

where A_1 is the gain or magnitude at passband edge frequency Ω_1

For small values of N the Chebyshev polynomial is given by

$$C_N(x) = \begin{cases} \cos(N\cos^{-1} x) & ; \text{ for } |x| \le 1 \\ \cosh(N\cosh^{-1} x) & ; \text{ for } |x| > 1 \end{cases}$$

For large values of N the Chebyshev polynomial is given by the recurrence relatioı ,

$$C_N(x) = 2xC_{N-1}(x) - C_{N-2}(x)$$

with initial values $C_0(x) = 1$ and $C_1(x) = x$

The transfer function of the analog system can be obtained from equation (4.53) by substituting Ω by s/j.

$$\therefore H_a(s)H(-s) = \frac{1}{1 + \epsilon^2 C_N^2\left(\frac{s/j}{\Omega_c}\right)}$$

For the normalized transfer function, let us replace s/Ω_c by s_n.

$$\therefore H_a(s_n)H(-s_n) = \frac{1}{1 + \epsilon^2 C_N^2(-js_n)}$$

For the transfer function of equation (4.58) we can determine 2N poles which are given by the roots of the denominator polynomial. It can be shown that the poles of the transfer function symmetrically lies on an ellipse in s-plane.

For a stable and causal filter the poles should lie on the left half of s-plane. Hence the desired filter transfer function is obtained by selecting N number of left half poles. When N is even all the poles are complex and exist as conjugate pair. When N is odd, one of the pole is real and all other poles are complex and exist as conjugate pair. Therefore the transfer function of chebyshev filters will be a product of second order factors. The analog filter transfer function of unnormalized chebyshev lowpass filter is given below.

Unnormalised chebyshev lowpass filter transfer function

Let N be the order of the filter.

When N is even, $H_a(s) = \prod_{k=1}^{\frac{N}{2}} \frac{B_k\Omega_c^2}{s^2 + b_k\Omega_c s + c_k\Omega_c^2}$

When N is odd, $H_a(s) = \frac{B_0\Omega_c}{s+c_0\Omega_c} \prod_{k=1}^{\frac{N-1}{2}} \frac{B_k\Omega_c^2}{s^2+b_k\Omega_c s+c_k\Omega_c^2}$

where, $b_k = 2y_N \sin\left(\frac{(2k-1)\pi}{2N}\right)$

$$c_k = y_N^2 + \cos^2\frac{(2k-1)\pi}{2N}$$

$$c_0 = y_N$$

$$y_N = \frac{1}{2}\left\{\left[\left(\frac{1}{\epsilon^2}+1\right)^{\frac{1}{2}}+\frac{1}{\epsilon}\right]^{\frac{1}{N}} - \left[\left(\frac{1}{\epsilon^2}+1\right)^{\frac{1}{2}}+\frac{1}{\epsilon}\right]^{-\frac{1}{N}}\right\}$$

For even values of N and unity dc gain filter, the parameter B_k are evaluated using the equation (4.65)

$$H_a(s)\Big|_{s=0} = \frac{1}{(1+\epsilon^2)^{\frac{1}{2}}}$$

For odd values of N and unity dc gain filter, the parameter B_k are evaluated using the equation (4.66)

$$H_a(s)\Big|_{s=0} = 1$$

While evaluating B_k's using equation (4.65) or (4.66), it is normal practice to take, $B_0 = B_1 = B_2 = \ldots\ldots = B_k$.

To determine the order N of the filter calculate a parameter N_1, using equation (4.67) and correct it to nearest integer. Then choose N such that $N \geq N_1$.

$$N_1 = \frac{\cosh^{-1}\left\{\frac{1}{\epsilon}\left(\frac{1}{A_2^2}-1\right)^{\frac{1}{2}}\right\}}{\cosh^{-1}\left(\frac{\Omega_2}{\Omega_1}\right)} \quad \ldots\ldots (4.67)$$

Frequency response of chebyshev filter

The frequency response of chebyshev filter depends on the order N as shown in fig 4.11. It can be observed that the approximated magnitude response approaches the ideal response

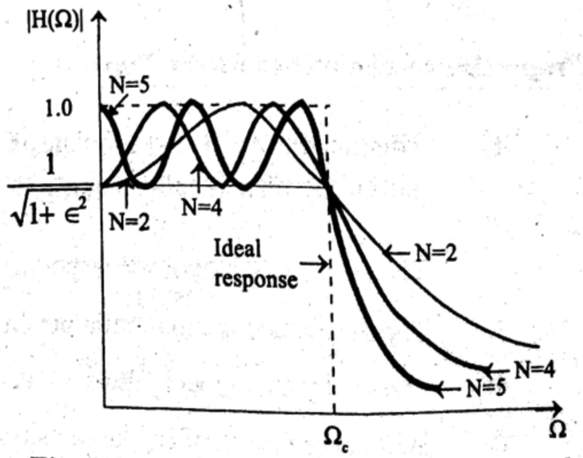


Fig 4.11. Magnitude response of Chebyshev type-1 lowpass filter for various values of N.

as the value of N increases. The magnitude response of Type-1 and Type-2 chebyshev filters are shown in fig 4.12.

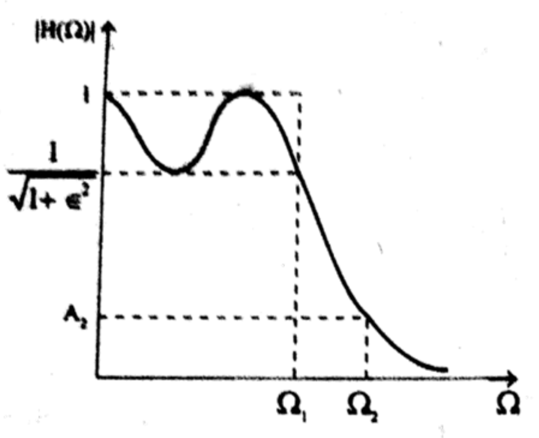


Fig a : Chebyshev Type - 1, when N is odd

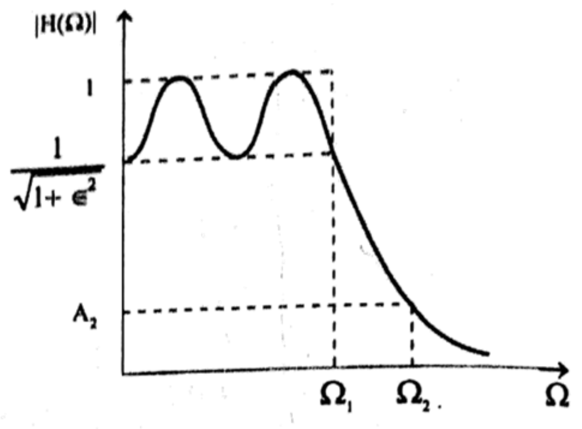


Fig b : Chebyshev Type - 1, when N is even

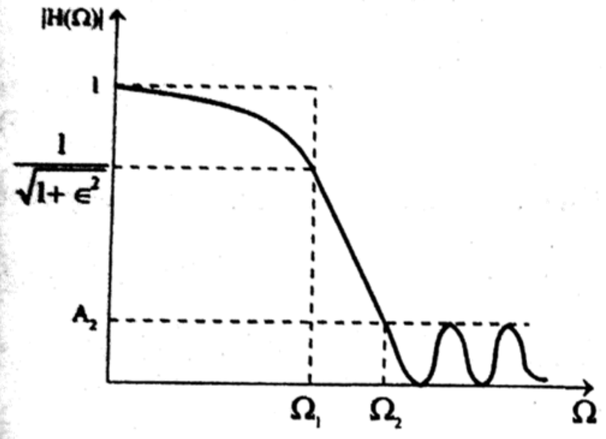


Fig c : Chebyshev Type - 2, when N is odd

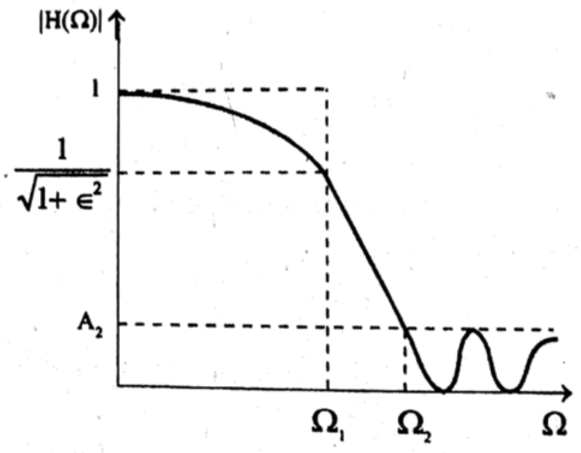


Fig d : Chebyshev Type - 2, when N is even

Fig 4.12 : Magnitude responses of analog chebyshev filters

Properties of chebyshev filters (Type-1)

1. The magnitude $|H_a(\Omega)|$ oscillates between 1 and $1/\sqrt{1+\epsilon^2}$ within the passband and so the filter is called equiripple in the passband.
2. The normalized magnitude response has a value of $1/\sqrt{1+\epsilon^2}$ at cutoff frequency Ω_c.
3. The magnitude is monotonic outside the passband.
4. The chebyshev type-1 filters are all pole designs.
5. With large values of N, the transition from passband to stopband becomes more sharp and approaches ideal characteristics.

Design procedure for lowpass digital chebyshev IIR filter

A_1 = Gain at a passband frequency ω_1.

A_2 = Gain at a stopband frequency ω_2.

Ω_1 = Analog frequency corresponding to ω_1.

Ω_2 = Analog frequency corresponding to ω_2.

1. Choose either bilinear or impulse invariant transformation.
2. Calculate the attenuation constant $\in$

$$\in = \left[\frac{1}{A_1^2} - 1\right]^{\frac{1}{2}}$$

3. Calculate the ratio $\frac{\Omega_2}{\Omega_1}$

For bilinear transformation, $\frac{\Omega_2}{\Omega_1} = \frac{\tan \omega_2 / 2}{\tan \omega_1 / 2}$

For impulse variant transformation, $\frac{\Omega_2}{\Omega_1} = \frac{\omega_2}{\omega_1}$

4. Decide the order N of the filter. Choose N such that $N \geq N_1$, where N_1 is given by

$$N_1 = \frac{\text{Cosh}^{-1}\left\{\frac{1}{\in}\left[\frac{1}{A_2^2} - 1\right]^{\frac{1}{2}}\right\}}{\text{Cosh}^{-1}\left(\frac{\Omega_2}{\Omega_1}\right)}$$

5. Calculate the analog cutoff frequency Ω_c.

For bilinear transformation, $\Omega_c = \frac{2}{T} \frac{\tan \frac{\omega_1}{2}}{\left[\frac{1}{A_1^2} - 1\right]^{\frac{1}{2N}}}$

For impulse invariant transformation, $\Omega_c = \frac{\omega_1 / T}{\left[\frac{1}{A_1^2} - 1\right]^{\frac{1}{2N}}}$

6. Determine the analog transfer function $H_a(s)$ of the filter.

When the order N is even, $H_a(s)$ is given by equation (4.74).

$$H_a(s) = \prod_{k=1}^{\frac{N}{2}} \frac{B_k\Omega_c^2}{s^2 + b_k\Omega_c s + c_k\Omega_c^2}$$

When the order N is odd, $H_a(s)$ is given by equation (4.75)

$$H_a(s) = \frac{B_0\Omega_c}{s + c_0\Omega_c} \prod_{k=1}^{\frac{N-1}{2}} \frac{B_k\,\Omega_c^2}{s^2 + b_k\Omega_c\,s + c_k\Omega_c^2}$$

Where, $b_k = 2y_N \sin\left(\frac{(2k-1)\pi}{2N}\right)$

$$c_k = y_N^2 + \text{Cos}^2\frac{(2k-1)\pi}{2N}$$

$$c_0 = y_N$$

$$y_N = \frac{1}{2}\left\{\left[\left(\frac{1}{\epsilon^2}+1\right)^{\frac{1}{2}} + \frac{1}{\epsilon}\right]^{\frac{1}{N}} - \left[\left(\frac{1}{\epsilon^2}+1\right)^{\frac{1}{2}} + \frac{1}{\epsilon}\right]^{-\frac{1}{N}}\right\}$$

For even values of N and unity dc gain filter, find B_k's such that

$$H_a(0) = \frac{1}{(1+\epsilon^2)^{\frac{1}{2}}}$$

For odd values of N and unity dc gain filter, find B_k's such that

$H_a(0) = 1$

(It is normal practice to take $B_0 = B_1 = B_2 \ldots\ldots\ldots = B_k$).

7. Using the chosen transformation, transform $H_a(s)$ to H(z), where H(z) is the transfer function of the digital filter.

8. Realize the digital filter transfer function H(z) by a suitable structure.

[Note : The highpass, bandpass and bandstop filters are obtained from lowpass filter design by frequency transformation.]

4.7 FREQUENCY TRANSFORMATIONS

The four basic types of filters are lowpass, highpass, bandpass and bandstop filters. In fig 4.13. the frequency response of the ideal cases are shown in solid lines and practical case in dotted lines.

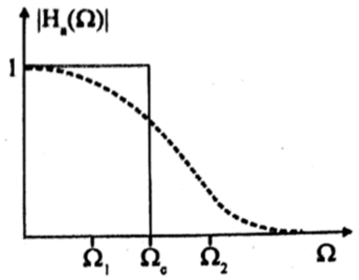


Fig a : Frequency response of lowpass filter

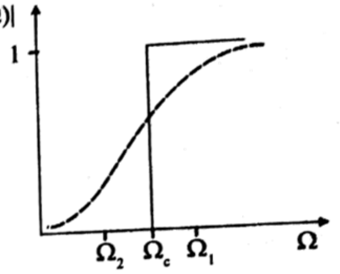


Fig b :Frequency response of highpass filter

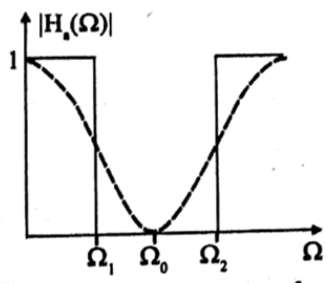


Fig c : Frequency response of bandstop filter

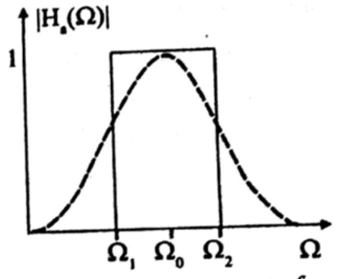


Fig d : Frequency response of bandpass filter

Fig 4.13 : Frequency response (magnitude response) of analog filters

The highpass or bandpass or bandstop filters are designed by designing a lowpass filter and then using frequency transformation, the transfer function of the desired filter is obtained. The frequency transformation can be carried in s-domain (Analog) or in z-domain (Digital).

Analog frequency transformation

Using analog frequency transformation the following filters can be designed.

1. Lowpass filter of another cutoff frequency.
2. Highpass filter with cutoff frequency Ω_c.
3. Bandpass filter with center frequency Ω_0 and Quality factor Q.
4. Bandstop filter with center frequency Ω_0 and Quality factor Q.

$$\text{where } \Omega_0 = \sqrt{\Omega_1\Omega_2} \quad \text{and} \quad Q = \frac{\Omega_0}{\Omega_2 - \Omega_1}$$

To design a filter, first design a lowpass filter from the given specifications, and determine the analog transfer function (either butterworth or chebychev transfer function) of the lowpass filter. Then choose the transformation from the table 4.1 and determine the analog transfer function of the desired filter.

Table 4.1

Filter Type	Transformation
Lowpass	$s \rightarrow \frac{s}{\Omega_c}$
Highpass	$s \rightarrow \frac{\Omega_c}{s}$
Bandpass	$s \rightarrow \frac{Q\left(s^2+\Omega_0^2\right)}{\Omega_0 s}$
Bandstop	$s \rightarrow \frac{\Omega_0 s}{Q\left(s^2+\Omega_0^2\right)}$

From the Analog transfer function $H_a(s)$ the digital transfer function H(z) is obtained by either bilinear transformation or impulse invariant transformation. The H(z) can be realized from a suitable structure.

EXAMPLE 4.8

The specification of the desired lowpass filter is

$$0.8 \le |H(\omega)| \le 1.0 \quad ; \quad 0 \le \omega \le 0.2\pi$$

$$|H(\omega)| \le 0.2 \quad ; \quad 0.32\pi \le \omega \le \pi$$

Design a) Butterworth digital filter using impulse invariant transformatio

b) Chebyshev digital filter using bilinear transformation.

SOLUTION

A) BUTTERWORTH DIGITAL FILTER

Given that, $A_1 = 0.8 \quad ; \quad \omega_1 = 0.2\pi$ **rad/sec**

$A_2 = 0.2 \quad ; \quad \omega_2 = 0.32\pi$ **rad/sec**

The transformation to be used is impulse invariant transformation.

For impulse invariant transformation, $\frac{\Omega_2}{\Omega_1} = \frac{\omega_2}{\omega_1} = \frac{0.32\pi}{0.2\pi} = 1.6$

$$N_1 = \frac{1}{2}\frac{\log\left\{\left(\frac{1}{A_2^2}-1\right)\Big/\left(\frac{1}{A_1^2}-1\right)\right\}}{\log\frac{\Omega_2}{\Omega_1}}$$

$$= \frac{1}{2}\frac{\log\left\{\left(\frac{1}{0.2^2}-1\right)\Big/\left(\frac{1}{0.8^2}-1\right)\right\}}{\log 1.6} = \frac{1}{2}\frac{\log\frac{24}{0.5625}}{\log 1.6} = \frac{1}{2}\frac{1.63}{0.2041} = 3.9931$$

Choose the order of the filter N such that $N \geq N_1$

Let the order of the filter, $N = 4$.

The analog cutoff frequency $\Omega_c = \dfrac{\omega_1 / T}{\left(\dfrac{1}{A_1^2}-1\right)^{\frac{1}{2N}}}$

Let $T = 1\,\text{sec}$

$$\therefore \Omega_c = \frac{\omega_1}{\left(\frac{1}{A_1^2}-1\right)^{\frac{1}{2N}}} = \frac{0.2\pi}{\left(\frac{1}{0.8^2}-1\right)^{\frac{1}{2\times 4}}} = \frac{0.2\pi}{0.9306} = 0.675 \text{ rad / sec}$$

The transfer function of the analog filter for even value of N is given by

$$H_a(s) = \prod_{k=1}^{\frac{N}{2}} \frac{\Omega_c^2}{s^2 + b_k\Omega_c s + \Omega_c^2}$$

where, $b_k = 2\sin\left(\dfrac{(2k-1)\pi}{2N}\right)$

Here $N = 4$, $\therefore$ $k = 1,2$

When k=1 ; $b_1 = 2\sin\left(\dfrac{1\times\pi}{2\times 4}\right) = 0.765$

When k=2 ; $b_2 = 2\sin\left(\dfrac{3\pi}{2\times 4}\right) = 1.848$

$$\therefore H_a(s) = \frac{\Omega_c^2}{s^2 + b_1\Omega_c s + \Omega_c^2} \times \frac{\Omega_c^2}{s^2 + b_2\Omega_c s + \Omega_c^2}$$

$$= \frac{(0.675)^2}{s^2 + (0.765\times 0.675)s + 0.675^2} \times \frac{(0.675)^2}{s^2 + (1.848\times 0.675)s + 0.675^2}$$

$$= \frac{0.2076}{(s^2 + 0.516s + 0.456)(s^2 + 1.247s + 0.456)}$$

The roots of the quadratic ($s^2+0.516s+0.456$) = 0, are

$$s = \frac{-0.516 \pm \sqrt{0.516^2 - 4\text{x}0.456}}{2} = -0.258 \pm j0.624 = -0.26 \pm j0.62$$

The roots of the quadratic ($s^2+1.247s+0.456$) = 0, are

$$s = \frac{-1.247 \pm \sqrt{1.247^2 - 4\text{x}0.456}}{2} = -0.623 \pm j0.259 = -0.62 \pm j0.26$$

$$\therefore H_a(s) = \frac{0.2076}{[s-(-0.26+j0.62)]\,[s-(-0.26-j0.62]\,[s-(-0.62+j0.26)]\,[s-(-0.62-j0.26)]}$$

By partial fraction expansion $H_a(s)$ can be expressed as,

$$H_a(s) = \frac{R_1}{s-(-0.26+j0.62)} + \frac{R_1^*}{s-(-0.26-j0.62)} + \frac{R_2}{s-(-0.62+j0.26)} + \frac{R_2^*}{s-(-0.62-j0.26)}$$

where R_1, R_1^*, R_2,R_2^* are residues.

$$R_1 = \frac{0.2076}{[s-(-0.26-j0.62)]\,[s-(-0.62+j0.26)]\,[s-(-0.62-j0.26)]}\Bigg|_{s=-0.26+j0.62}$$

$$= \frac{0.2076}{[-0.26+j0.62+0.26+j0.62]\,[-0.26+j0.62+0.62-j0.26]\,[-0.26+j0.62+0.62+j0.26]}$$

$$= \frac{0.2076}{j1.24(0.36+j0.36)(0.36+j0.88)} = \frac{0.2076}{1.24\angle 1.57 \times 0.5091\angle 0.785 \times 0.95\angle 1.182}$$

$$= \frac{0.2076}{1.24 \times 0.5091 \times 0.95}\angle -1.57 - 0.785 - 1.182$$

$$= 0.346\angle -3.537 = -0.32 + j0.13$$

***Note** : The complex number in the rectangular form is converted to polar form (and viceversa) using a calculator in radians mode.*

$$R_1^* = \text{Conjugate of } R_1 = -0.32 - j0.13$$

$$R_2 = \frac{0.2076}{[s-(-0.26+j0.62)]\,[s-(-0.26-j0.62)]\,[s-(-0.62-j0.26)]}\Bigg|_{s=-0.62+j0.26}$$

$$= \frac{0.2076}{(-0.62+j0.26+0.26-j0.62)(-0.62+j0.26+0.26+j0.62)(-0.62+j0.26+0.62+j0.26)}$$

$$= \frac{0.2076}{(-0.36-j0.36)(-0.36+j0.88)j0.52} = \frac{0.2076}{0.5091\angle -2.356 \times 0.9508\angle 1.959 \times 0.52\angle 1.57}$$

$$= \frac{0.2076}{0.5091 \times 0.9508 \times 0.52} \angle +2.356 - 1.959 - 1.57 = 0.8248\angle -1.173 = 0.32 - j0.76$$

$R_2^* =$ **Conjugate of** $R_2 = 0.32 + j0.76$

$$\therefore H_a(s) = \frac{-0.32 + j0.13}{s-(-0.26+j0.62)} + \frac{-0.32 - j0.13}{s-(-0.26-j0.62)} + \frac{0.32 - j0.76}{s-(-0.62+j0.26)} + \frac{0.32 + j0.76}{s-(-0.62-j0.26)}$$

EXAMPLE 4.10

The specification of the desired lowpass digital filter is

$0.9 \le |H(\omega)| \le 1.0 \quad ; \quad 0 \le \omega \le 0.25\pi$

$|H(\omega)| \le 0.24 \quad ; \quad 0.5\pi \le \omega \le \pi$

Design a chebyshev digital filter using impulse invariant transformation.

SOLUTION

Given that, $A_1 = 0.9 \quad ; \quad \omega_1 = 0.25\pi$

$A_2 = 0.24 \quad ; \quad \omega_2 = 0.5\pi$

The transformation to be used is impulse invariant transformation.

For impulse invariant transformation, $\dfrac{\Omega_2}{\Omega_1} = \dfrac{\omega_2}{\omega_1} = \dfrac{0.5\pi}{0.25\pi} = 2$

The attenuation constant, $\in = \left(\dfrac{1}{A_1^2} - 1\right)^{\frac{1}{2}} = \left(\dfrac{1}{0.9^2} - 1\right)^{\frac{1}{2}} = 0.484$

$$N_1 = \frac{\cosh^{-1}\left\{\dfrac{1}{\in}\left(\dfrac{1}{A_2^2} - 1\right)^{\frac{1}{2}}\right\}}{\cosh^{-1}\dfrac{\Omega_2}{\Omega_1}} = \frac{\cosh^{-1}\left\{\dfrac{1}{0.484}\left(\dfrac{1}{0.24^2} - 1\right)^{\frac{1}{2}}\right\}}{\cosh^{-1} 2}$$

$$= \frac{\cosh^{-1} 8.357}{\cosh^{-1} 2} = \frac{2.813}{1.317} = 2.136 \approx 3$$

Choose N such that, $N \ge N_1$. Let the order of the filter, $N = 3$. Also let $T = 1$ sec.

The analog cutoff frequency, $\Omega_c = \dfrac{\omega_1/T}{\left(\dfrac{1}{A_1^2} - 1\right)^{\frac{1}{2N}}} = \dfrac{0.25\pi}{\left(\dfrac{1}{0.9^2} - 1\right)^{\frac{1}{2\times 3}}} = \dfrac{0.25\pi}{(0.235)^{\frac{1}{6}}} = 1$ rad / sec.

The transfer function of the analog filter for odd values of N is given by,

$$H_a(s) = \frac{B_0\Omega_c}{s + c_0\Omega_c} \prod_{k=1}^{\frac{N-1}{2}} \frac{B_k\Omega_c^2}{s^2 + b_k\Omega_c s + c_k\Omega_c^2}$$

Here $N = 3$, $\therefore k = 1$

$$\therefore H_a(s) = \frac{B_0\Omega_c}{s + c_0\Omega_c} \times \frac{B_1\Omega_c^2}{s^2 + b_1\Omega_c s + c_1\Omega_c^2}$$

$$y_N = \frac{1}{2}\left\{\left[\left(\frac{1}{\epsilon^2}+1\right)^{\frac{1}{2}} + \frac{1}{\epsilon}\right]^{\frac{1}{N}} - \left[\left(\frac{1}{\epsilon^2}+1\right)^{\frac{1}{2}} + \frac{1}{\epsilon}\right]^{-\frac{1}{N}}\right\}$$

$$= \frac{1}{2}\left\{\left[\left(\frac{1}{0.484^2}+1\right)^{\frac{1}{2}} + \frac{1}{0.484}\right]^{\frac{1}{3}} - \left[\left(\frac{1}{0.484^2}+1\right)^{\frac{1}{2}} + \frac{1}{0.484}\right]^{-\frac{1}{3}}\right\}$$

$$= \frac{1}{2}\{1.634 - 0.612\} = 0.511$$

$$c_0 = y_N = 0.511$$

$$c_k = y_N^2 + \cos^2\frac{(2k-1)\pi}{2N}$$

When $k = 1$, $c_1 = y_N^2 + \cos^2\left(\frac{\pi}{2\times 3}\right) = 0.511^2 + \cos^2\left(\frac{\pi}{6}\right) = 1.011$

$$b_k = 2y_N \sin\frac{(2k-1)\pi}{2N}$$

When $k = 1$, $b_1 = 2y_N \sin\left(\frac{\pi}{2\times 3}\right) = 2\times 0.511 \times \sin\left(\frac{\pi}{6}\right) = 0.511$

$$\therefore H_a(s) = \frac{B_0}{s+0.511} \times \frac{B_1}{s^2 + 0.511s + 1.011}$$

When $s = 0$, $H_a(0) = \frac{B_0 B_1}{0.511 \times 1.011} = 1.936 B_0 B_1$

Let $H_a(0) = 1$, $\therefore$ $1.936\ B_0 B_1 = 1$

Let $B_0 = B_1$, $\therefore B_0^2 = \frac{1}{1.936}$ or $B_0 = \frac{1}{\sqrt{1.936}} = 0.719$

$\therefore B_0 = B_1 = 0.719$

$$\therefore H_a(s) = \frac{0.719^2}{(s+0.511)(s^2+0.511s+1.011)} = \frac{0.517}{(s+0.511)(s^2+0.511s+1.011)} \quad \text{..... (4.10.1)}$$

By partial fraction expansion $H_a(s)$ can be expressed as,

$$H_a(s) = \frac{0.517}{(s+0.511)(s^2+0.511s+1.011)} = \frac{A}{s+0.511} + \frac{Bs+C}{s^2+0.511s+1.011} \quad \text{.....(4.10.2)}$$

On cross multiplying the equation (4.10.2) we get

$$0.517 = A(s^2 + 0.511s + 1.011) + (Bs + C)(s + 0.511)$$

$$0.517 = As^2 + 0.511As + 1.011A + Bs^2 + 0.511Bs + Cs + 0.511C \quad \text{..... (4.10.3)}$$

On equating the coefficient of s^2 in equation (4.10.3) we get

$$A + B = 0$$

On equating the coefficients of s in equation (4.10.3) we get

$$0.511A + 0.511B + C = 0$$

$$0.511(A+B) + C = 0$$

On equating the constants in equation (4.10.3) we get

$$1.011A + 0.511C = 0.517$$

From equations (4.10.4) & (4.10.5), we get, $C = 0$.

On substituting, $C = 0$, in equation (4.10.6), we get, $A = 0.517/1.011 = 0.511$

From equation (4.10.4) we get, $B = -A = -0.511$.

$$\therefore H_a(s) = \frac{A}{s+0.511} + \frac{Bs+C}{s^2+0.511s+1.011} = \frac{0.511}{s+0.511} - \frac{0.511s}{s^2+0.511s+1.011}$$

$$= \frac{0.511}{s+0.511} - \frac{0.511s}{\left(s^2+2\times 0.256s+0.256^2\right)+\left(\sqrt{1.011-0.256^2}\right)^2}$$

$$= \frac{0.511}{s+0.511} - 0.511\frac{s+0.256-0.256}{(s+0.256)^2+0.972^2}$$

$$= \frac{0.511}{s+0.511} - 0.511\frac{s+0.256}{(s+0.256)^2+0.972^2} + 0.511\frac{0.256}{(s+0.256)^2+0.972^2}$$

$$= \frac{0.511}{s+0.511} - 0.511\frac{s+0.256}{(s+0.256)^2+0.972^2} + \frac{0.511\times 0.256}{0.972}\frac{0.972}{(s+0.256)^2+0.972^2}$$

$$= \frac{0.511}{s+0.511} - 0.511\frac{s+0.256}{(s+0.256)^2+0.972^2} + 0.135\frac{0.972}{(s+0.256)^2+0.972^2}$$

2. 5 **Direct Form Structures**

The output signal y[k]=H{x[k]}y[k]=H{x[k]} of a recursive linear-time invariant (LTI) system and the computational realization of above equation requires additions, multiplications, the actual and past samples of the input signal x[k]x[k], and the past samples of the output signal y[k]y[k]. Technically this can be realized by

- adders
- multipliers, and
- unit delays or storage elements.

These can be arranged in different topologies. A certain class of structures, which is introduced in the following, is known as *direct form structures*. Other known forms are for instance *cascaded sections*, *parallel* sections, lattice structures and state-space structures.

For the following it is assumed that a0=1a0=1. This can be achieved for instance by dividing the remaining coefficients by a0a0.

2.5.1 Direct Form I

The direct form I is derived by rearranging the difference equation.It is now evident that we can realize the recursive filter by a superposition of a non-recursive and a recursive part. With the elements given above, this results in the following block-diagram

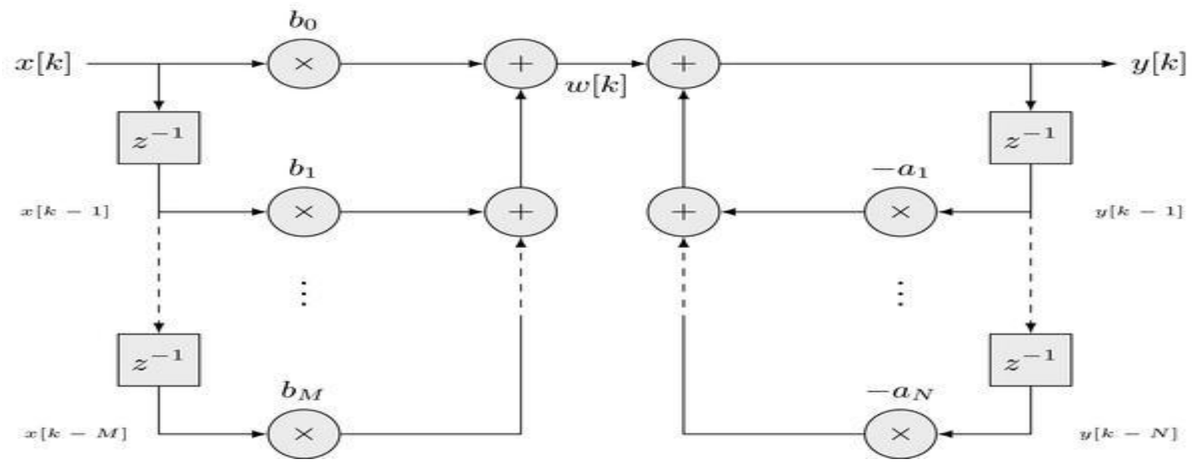
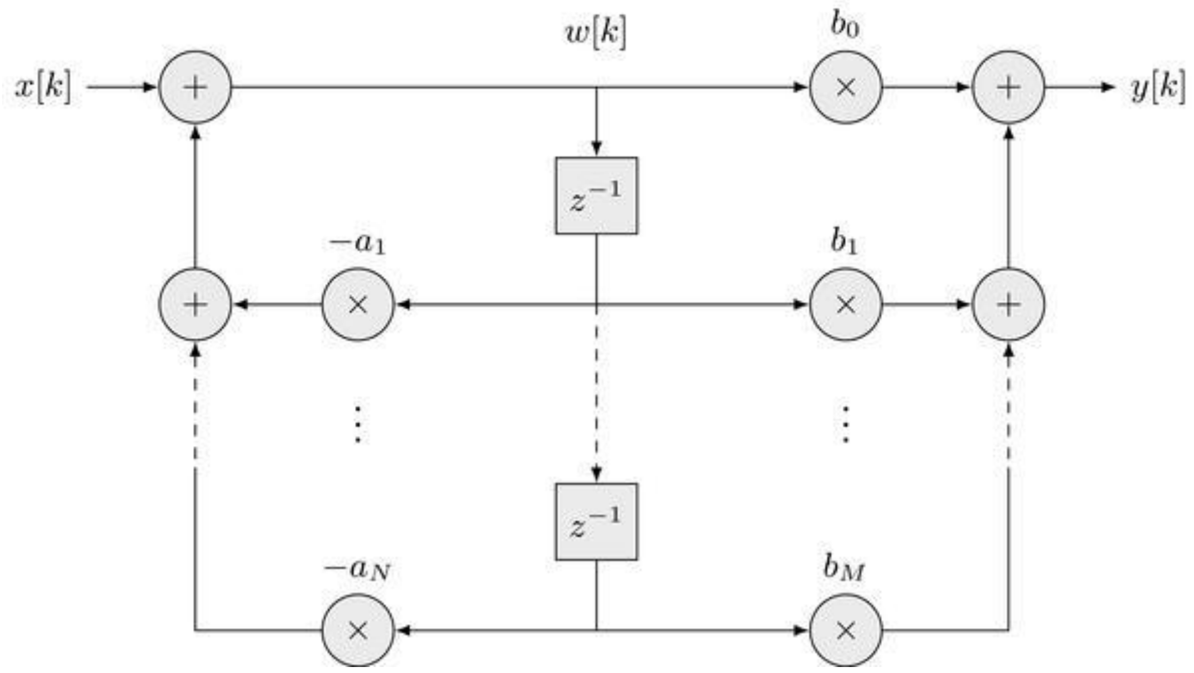


Fig 2.10 Direct form I filter

This representation is not canonical since N+MN+M unit delays are required to realize a system of order NN. A benefit of the direct form I is that there is essentially only one summation point which has to be taken care of when considering quantized variables and overflow. The output signal y[k]y[k] for the direct form I is computed by realizing above equation.The block diagram of the direct form I can be interpreted as the cascade of two systems. Denoting the signal in between both as w[k]w[k] and discarding initial values we getwhere h1[k]=[b0,b1,…,bM]h1[k]=[b0,b1,…,bM] denotes the impulse response of the non-recursive part and h2[k]=[1,−a1,…,−aN]h2[k]=[1,−a1,…,−aN] for the recursive part. From the last equality of the second equation and the commutativity of the convolution it becomes clear that the order of the cascade can be exchanged.

2.5.2 Direct Form II

The direct form II is yielded by exchanging the two systems in above block diagram and noticing that there are two parallel columns of delays which can be combined, since they are redundant. For N=MN=M it is given as



2.11 Direct form II filter

Other cases with N≠MN≠M can be considered for by setting coefficients to zero. This form is a canonical structure since it only requires NN unit delays for a recursive filter of order NN. The output signal y[k]y[k] for the direct form II is computed by the following equations The samples w[k−m]w[k−m] are termed *state* (variables) of a digital filter.

2.5.3 CASCADE FORM STRUCTURE FOR IIR SYSTEMS

In cascade form, stages are cascaded (connected) in series. The output of one system is input to another. Thus total K numbers of stages are cascaded. The total system function'H' is given**.**

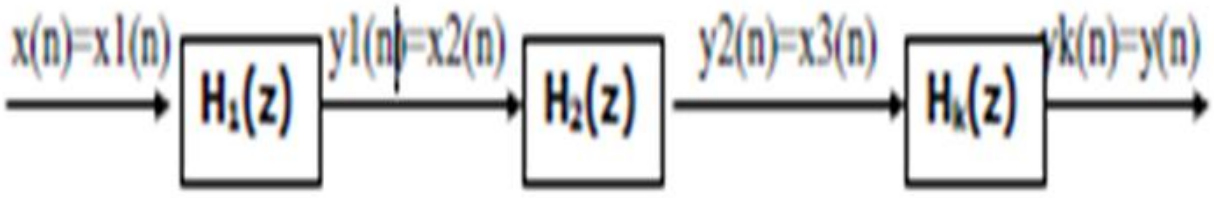


Fig 2.12 Cascade realization

2.5.4 PARALLEL FORM STRUCTURE FOR IIR SYSTEMS

In parallel form of realization, the system has one input and the output is obtained by adding the outputs from the sub systems

System function for IIR systems is given as

$$H(z) = \sum_{K=0}^{M} b_k z^{-k} / 1 + \sum_{k=1}^{N} a_k z^{-k}$$

$$= b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots\ldots\ldots + b_M z^{-M} / 1 + a_1 z^{-1} + a_2 z^{-2} + \ldots\ldots + a_N z^{-N}$$

The above system function can be expanded in partial fraction as follows

$$H(z) = C + H_1(z) + H_2(z) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots + H_k(z)$$

Where C is constant and Hk(z) is given as

$$Hk(z) = b_{k0} + b_{k1} z^{-1} / 1 + a_{k1} z^{-1} + a_{k2} z^{-2}$$

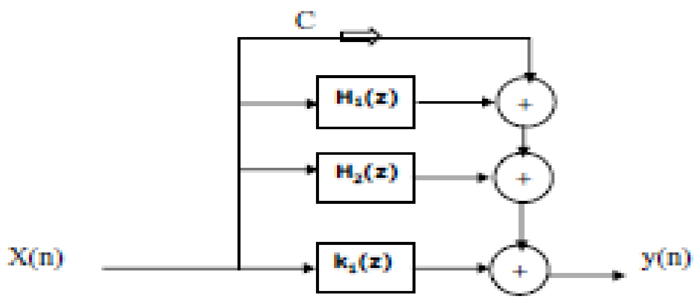


Fig 2.13 Parallel form of realization

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL AND ELECTRONICS

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

UNIT – III - – DESIGN OF FINITE IMPULSE RESPONSE FILTER-SEC1315

FINITE IMPULSE RESPONSE DIGITAL FILTERS

3.1 Symmetric and Anti symmetric FIR filters

FIR filters are digital filters with finite impulse response. They are also known as **non-recursive digital filters** as they do not have the feedback (a recursive part of a filter), even though recursive algorithms can be used for FIR filter realization. FIR filters can be designed using different methods, but most of them are based on ideal filter approximation. The objective is not to achieve ideal characteristics, as it is impossible anyway, but to achieve sufficiently good characteristics of a filter. The transfer function of FIR filter approaches the ideal as the filter order increases, thus increasing the complexity and amount of time needed for processing input samples of a signal being filtered. The resulting frequency response can be a monotone function or an oscillatory function within a certain frequency range. The waveform of frequency response depends on the method used in design process as well as on its parameters.

This book describes the most popular method for FIR filter design that uses window functions. The characteristics of the transfer function as well as its deviation from the ideal frequency response depend on the filter order and window function in use.

Each filter category has both advantages and disadvantages. This is the reason why it is so important to carefully choose category and type of a filter during design process.

FIR filters can have linear phase characteristic, which is not like IIR filters that will be discussed in Chapter 3. Obviously, in such cases when it is necessary to have a linear phase characteristic, FIR filters are the only option available. If the linear phase characteristic is not necessary, as is the case with processing speech signals, FIR filters are not good solution at all.

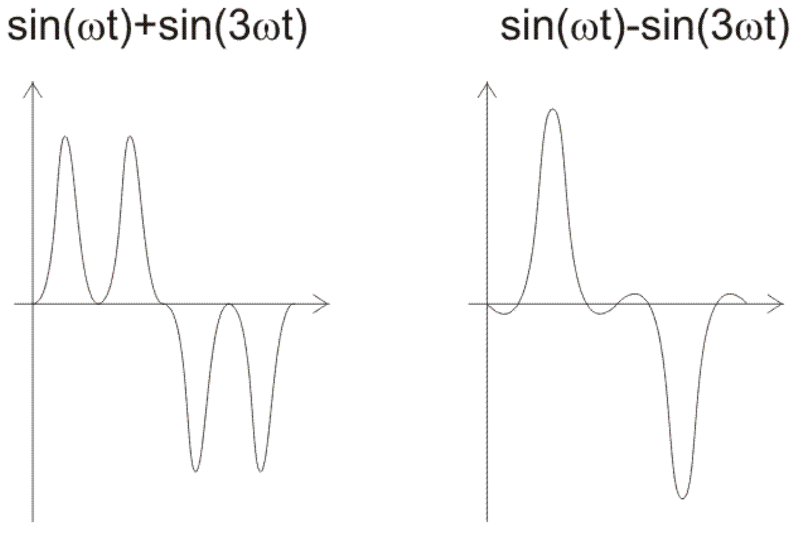


***Fig.3.1. Illustration of input and output signals of non-linear phase systems*.**

The system introduces a phase shift of 0 radians at the frequency of ω, and π radians at three times that frequency. Input signal consists of natural frequency ω and one harmonic with the same amplitude at three times that frequency. Figure 2-1-3. shows the block diagram of input signal (left) and output signal (right). It is obvious that these two signals have different waveforms. The power of signals is not changed, nor the amplitudes of harmonics, only the phase of the second harmonic is changed.

If we assume that the input is a speech signal whose phase characteristic is not of the essence, such distortion in the phase of the signal would be unimportant. In this case, the system satisfies all necessary requirements. However, if the phase characteristic is of importance, such a great distortion mustn't be allowed.

In order that the phase characteristic of a FIR filter is linear, the impulse response must be symmetric or anti-symmetric, which is expressed in the following way:

h[n] = h[N-n-1] ; symmetric impulse response (about its middle element)

h[n] = -h[N-n-1] ; anti-symmetric impulse response (about its middle element)

One of the drawbacks of FIR filters is a high order of designed filter. The order of FIR filter is remarkably higher compared to an IIR filter with the same frequency response. This is the reason why it is so important to use FIR filters only when the linear phase characteristic is very important.

A number of delay lines contained in a filter, i.e. a number of input samples that should be saved for the purpose of computing the output sample, determines the order of a filter. For example, if the filter is assumed to be of order 10, it means that it is necessary to save 10 input samples preceeding the current sample. All eleven samples will affect the output sample of FIR filter.

The transform function of a typical FIR filter can be expressed as a polynomial of a complex variable z^{-1}. All the poles of the transfer function are located at the origin. For this reason, FIR filters are guaranteed to be stable, whereas IIR filters have potential to become unstable.

3.2. Finite impulse response (FIR) filter design methods

Most FIR filter design methods are based on ideal filter approximation. The resulting filter approximates the ideal characteristic as the filter order increases, thus making the filter and its implementation more complex.

The filter design process starts with specifications and requirements of the desirable FIR filter. Which method is to be used in the filter design process depends on the filter specifications and implementation. This chapter discusses the FIR filter design method using window functions.

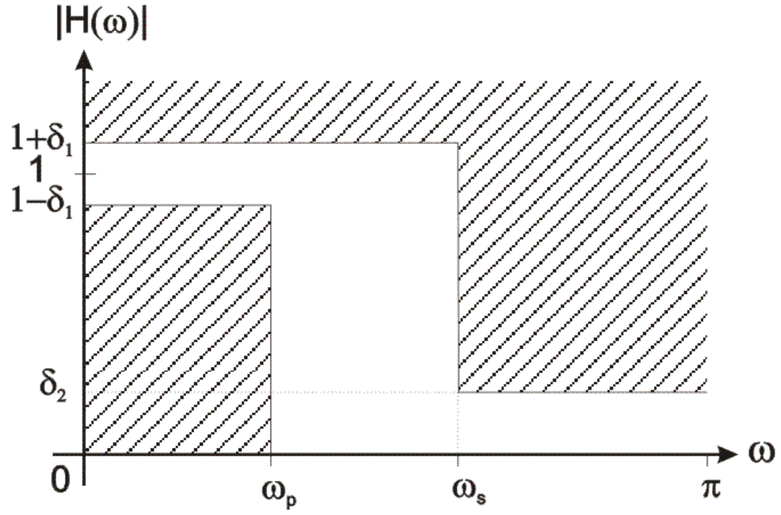
Each of the given methods has its advantages and disadvantages. Thus, it is very important to carefully choose the right method for FIR filter design. Due to its simplicity and efficiency, the window method is most commonly used method for designing filters. The sampling frequency method is easy to use, but filters designed this way have small attenuation in the stopband.

As we have mentioned above, the design process starts with the specification of desirable FIR filter.

3.2.1. *Basic concepts and FIR filter specification*

First of all, it is necessay to learn the basic concepts that will be used further in this book. You should be aware that without being familiar with these concepts, it is not possible to understand analyses and synthesis of digital filters.

Figure 3.2 illustrates a low-pass digital filter specification. The word specification actually refers to the frequency response specification.



•

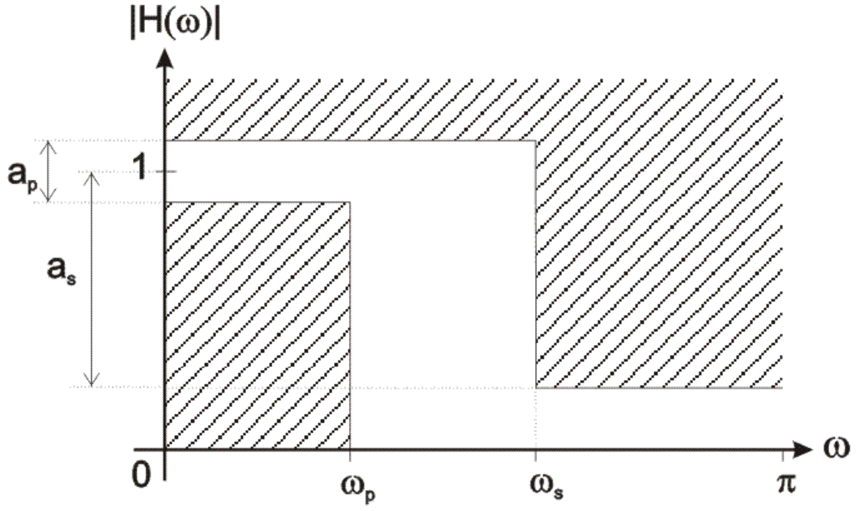


Fig.3.2. A low-pass digital filter specification

- ωp – normalized cut-off frequency in the passband;
- ωs – normalized cut-off frequency in the stopband;
- δ1 – maximum ripples in the passband;
- δ2 – minimum attenuation in the stopband **[dB]**;
- ap – maximum ripples in the passband; and
- as – minimum attenuation in the stopband **[dB]**.

$$a_p = 20\log_{10}\left(\frac{1+\delta_1}{1-\delta_1}\right)$$

$$a_s = -20\log_{10}\delta_2$$

Frequency normalization can be expressed as follows:

$$\omega = \frac{2\pi f}{f_s}$$

where:

- fs is a sampling frequency;
- f is a frequency to normalize; and
- ω is normalized frequency.

Table.3.1.Filters

Type of filter	Frequency response $h_d[n]$
low-pass filter	$h_d[n] = \begin{cases} \dfrac{\sin[\omega_c(n-M)]}{\pi(n-M)}; & n \neq M \\ \dfrac{\omega_c}{\pi}; & n = M \end{cases}$
high-pass filter	$h_d[n] = \begin{cases} 1-\dfrac{\omega_c}{\pi}; & n \neq M \\ -\dfrac{\sin(\omega_c(n-M))}{\pi(n-M)}; & n = M \end{cases}$
band-pass filter	$h_d[n] = \begin{cases} \dfrac{\sin(\omega_{c2}(n-M))}{\pi(n-M)} - \dfrac{\sin(\omega_{c1}(n-M))}{\pi(n-M)}; & n \neq M \\ \dfrac{\omega_{c2}-\omega_{c1}}{\pi}; & n = M \end{cases}$
band-stop filter	$h_d[n] = \begin{cases} \dfrac{\sin(\omega_{c1}(n-M))}{\pi(n-M)} - \dfrac{\sin(\omega_{c2}(n-M))}{\pi(n-M)}; & n \neq M \\ 1-\dfrac{\omega_{c2}-\omega_{c1}}{\pi}; & n = M \end{cases}$

The value of variable **n** ranges between 0 and N, where N is the filter order. A constant M can be expressed as M = N / 2. Equivalently, N can be expressed as N = 2M.

The constant M is an integer if the filter order N is even, which is not the case with odd order filters. If M is an integer (even filter order), the ideal filter frequency response is symmetric about its Mth sample which is found via expression shown in the table 2-2-1 above. If M is not an integer, the ideal filter frequency response is still symmetric, but not about some frequency response sample.

Since the variable **n** ranges between 0 and N, the ideal filter frequency response has N+1 sample.

If it is needed to find frequency response of a non-standard ideal filter, the expression for inverse Fourier transform must be used:

$$h_d[n] = \frac{1}{\pi}\int_0^{\pi} e^{j\omega(n-M)}d\omega$$

Non-standard filters are rarely used. However, if there is a need to use some of them, the integral above must be computed via various numerical methodes.

3..3 FIR filter design using window functions

The FIR filter design process via window functions can be split into several steps:

1. Defining filter specifications;
2. Specifying a window function according to the filter specifications;
3. Computing the filter order required for a given set of specifications;
4. Computing the window function coefficients;
5. Computing the ideal filter coefficients according to the filter order;
6. Computing FIR filter coefficients according to the obtained window function and ideal filter coefficients;
7. If the resulting filter has too wide or too narrow transition region, it is necessary to change the filter order by increasing or decreasing it according to needs, and after that steps 4, 5 and 6 are iterated as many times as needed.

The final objective of defining filter specifications is to find the desired normalized frequencies (ωc, ωc1, ωc2), transition width and stopband attenuation. The window function and filter order are both specified according to these parameters.

Accordingly, the selected window function must satisfy the given specifications. After this step, that is, when the window function is known, we can compute the filter order required for a given set of specifications. When both the window function and filter order are known, it is possible to calculate the window function coefficients w[n] using the formula for the specified window function.

1.Rectangular Window The rectangular window is what you would obtain if you were to simply segment a finite portion of the impulse response without any shaping in the time domain:

$$\mathrm{w(n)} = 1 \quad 0 \le \mathrm{n} \le \mathrm{M}\,,$$

$$= 0 \text{ otherwise}$$

2.Bartlett (or triangular) window

The Bartlett window is triangularly shaped:

$$w(n) = 1 - \frac{2n}{M} - 1 \quad 0 \leq n \leq M\,,$$

$$=0 \text{ otherwise}$$

3.Hanning window

The Hanning window(or more properly, the von Hann window) is nothing more than a raised cosine:

$$w(n) = 0.5 - 0.5\cos\left(\frac{2\pi n}{M}\right) \quad 0 \leq n \leq M\,,$$

$$=0 \text{ otherwise}$$

4. Hamming window

$$w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{M}\right) \quad 0 \leq n \leq M\,,$$

$$=0 \text{ otherwise}$$

5.Blackmam window

The Hanning and Hamming have a constant and a cosine term; the Blackman window adds a cosine at twice the frequency

$$w(n) = 0.42 - 0.5\cos\left(\frac{2\pi n}{M}\right) + 0.08\cos\left(\frac{4\pi n}{M}\right) \quad 0 \leq n \leq M\,,$$

$$=0 \text{ otherwise}$$

After estimating the window function coefficients, it is necessary to find the ideal filter frequency samples. The expressions used for computing these samples are discussed in section 2.2.3 under **Ideal filter approximation**. The final objective of this step is to obtain the coefficients $h_d[n]$. Two sequencies w[n] and $h_d[n]$ have the same number of elements.

The next step is to compute the frequency response of designed filter h[n] using the following expression:

$$h[n] = w[n] \cdot h_d[n]$$

Lastly, the transfer function of designed filter will be found by transforming impulse response via Fourier transform:

$$H(e^{j\omega}) = \sum_{n=0}^{N} h[n] \cdot e^{-jn\omega}$$

or via Z-transform:

$$H(z) = \sum_{n=0}^{N} h[n] z^{-n}$$

If the transition region of designed filter is wider than needed, it is necessary to increase the filter order, reestimate the window function coefficients and ideal filter frequency samples, multiply them in order to obtain the frequency response of designed filter and reestimate the transfer function as well. If the transition region is narrower than needed, the filter order can be decreased for the purpose of optimizing hardware and/or software resources. It is also necessary to reestimate the filter frequency coefficients after that.

PROBLEMS

Use the window design method to design a linear phase FIR filter of order *N* = 24 to approximate the following ideal frequency response magnitude

$$|H_d(e^{j\omega})| = \begin{cases} 1 & |\omega| \leq 0.2\pi \\ 0 & 0.2\pi < |\omega| \leq \pi \end{cases}$$

The ideal filter that we would like to approximate is a low-pass filter with a cutoff frequency = 0.2. With *N* = 24, the frequency response of the filter that is to be designed has the form

$$H(e^{j\omega}) = \sum_{n=0}^{24} h(n) e^{-jn\omega}$$

Therefore, the delay of $h(n)$ is $= N/2 = 12$, and the ideal unit sample response that is to be windowed is

$$h_d(n) = \frac{\sin[0.2\pi(n-12)]}{(n-12)\pi}$$

All that is left to do in the design is to select a window. With the length of the window fixed, there is a trade-off between the width of the transition band and the amplitude of the passband and stopband ripple. With a rectangular window, which provides the smallest transition band,

$$\Delta\omega = 2\pi \cdot \frac{0.9}{24} = 0.075\pi$$

and the filter is

$$h(n) = \begin{cases} \dfrac{\sin[0.2\pi(n-12)]}{(n-12)\pi} & 0 \le n \le 24 \\ 0 & \text{otherwise} \end{cases}$$

However, the stopband attenuation is only 21 dB, which is equivalent to a ripple of 0.089. With a Hamming window, on the other hand,

$$h(n) = \left[0.54 - 0.46\cos\left(\frac{2\pi n}{24}\right)\right] \cdot \frac{\sin[0.2\pi(n-12)]}{(n-12)\pi} \qquad 0 \le n \le 24$$

and the stopband attenuation is 53 dB, or ? $_s$ = 0.0022. However, the width of the transition band increases to

$$\Delta\omega = 2\pi \cdot \frac{3.3}{24} = 0.275\pi$$

which, for most designs, would be too wide.

3..4 .Frequency sampling method:

The frequency sampling method allows us to design recursive and nonrecursive FIR filters for both standard frequency selective and filters with arbitrary frequency response. A. No recursive frequency sampling filters : The problem of FIR filter design is to find a finite–length impulse response h (n) that corresponds to desired frequency response. In this method h (n) can be determined by uniformly sampling, the desired frequency response H_D (ω) at the N points and finding its inverse DFT of the frequency samples.

Problem

EXAMPLE .

Determine the coefficients of a linear-phase FIR filter of length N=15 which has a symmetric unit sample response and a frequency response that satisfies the conditions

$$H\left(\frac{2\pi k}{15}\right) = \begin{cases} 1 & ; \text{ for } k = 0,\ 1,\ 2,\ 3 \\ 0.4 & ; \text{ for } k = 4 \\ 0 & ; \text{ for } k = 5,\ 6,\ 7 \end{cases}$$

SOLUTION

For linear phase FIR filter the phase function, $\theta(\omega) = -\alpha\omega$, where $\alpha = (N-1)/2$.
Here N = 15, $\therefore \alpha = (15-1)/2 = 7$.

Also, here $\omega = \omega_k = \frac{2\pi k}{N} = \frac{2\pi k}{15}$. Hence we can go for type-1 design.

In this problem the samples of the magnitude response of the ideal (desired) filter are directly given for various values of k.

$$\therefore \tilde{H}(k) = H_d(\omega)\Big|_{\omega=\omega_k} = \begin{cases} 1\, e^{-j\alpha\omega_k} & ; k = 0,\ 1,\ 2,\ 3 \\ 0.4\, e^{-j\alpha\omega_k} & ; k = 4 \\ 0 & ; k = 5,\ 6,\ 7 \end{cases}$$

where, $\omega_k = \frac{2\pi k}{15}$

When k = 0, $\tilde{H}(0) = e^{-j\alpha\omega_k} = e^{-j7\times\frac{2\pi\times 0}{15}} = 1$

When k = 1, $\tilde{H}(1) = e^{-j\alpha\omega_k} = e^{-j7\times\frac{2\pi\times 1}{15}} = e^{-j\frac{14\pi}{15}}$

When k = 2, $\tilde{H}(2) = e^{-j\alpha\omega_k} = e^{-j7\times\frac{2\pi\times 2}{15}} = e^{-j\frac{28\pi}{15}}$

When k = 3, $\tilde{H}(3) = e^{-j\alpha\omega_k} = e^{-j7\times\frac{2\pi\times 3}{15}} = e^{-j\frac{42\pi}{15}}$

When k = 4, $\tilde{H}(4) = 0.4\, e^{-j\alpha\omega_k} = 0.4\, e^{-j7\times\frac{2\pi\times 4}{15}} = 0.4\, e^{-j\frac{56\pi}{15}}$

When k = 5, $\tilde{H}(5) = 0$

When k = 6, $\tilde{H}(6) = 0$

When k = 7, $\tilde{H}(7) = 0$

The samples of impulse response are given by

$$h(n) = \frac{1}{N}\left\{\tilde{H}(0) + 2\sum_{k=1}^{\frac{N-1}{2}} \text{Re}\left[\tilde{H}(k)\, e^{j2\pi nk/N}\right]\right\}$$

$$= \frac{1}{15}\left\{\tilde{H}(0) + 2\sum_{k=1} \text{Re}\left[\tilde{H}(k)\, e^{j2\pi nk/15}\right]\right\}$$

$$= \frac{1}{15}\left\{\tilde{H}(0) + 2\sum_{k=1}^{3} \text{Re}\left[\tilde{H}(k)\, e^{j2\pi nk/15}\right] + 2\ \text{Re}\left[\tilde{H}(4)\, e^{j2\pi n4/15}\right]\right\}$$

$$= \frac{1}{15}\left\{1 + 2\sum_{k=1}^{3} \text{Re}\left[e^{-j7\times\frac{2\pi k}{15}} \times e^{\frac{j2\pi nk}{15}}\right] + 2\ \text{Re}\left[0.4\, e^{-j7\frac{2\pi\times 4}{15}} \times e^{+j\frac{2\pi n4}{15}}\right]\right\}$$

$$= \frac{1}{15}\left\{1 + 2\sum_{k=1}^{3} \text{Re}\left[e^{\frac{j2\pi k}{15}(n-7)}\right] + 2\ \text{Re}\left[0.4\, e^{\frac{j8\pi}{15}(n-7)}\right]\right\}$$

$$= \frac{1}{15}\left\{1 + 2\sum_{k=1}^{3} \cos\frac{2\pi k}{15}(n-7) + 0.8\cos\frac{8\pi}{15}(n-7)\right\}$$

$$= \frac{1}{15} + \frac{2}{15}\cos\frac{2\pi}{15}(n-7) + \frac{2}{15}\cos\frac{4\pi}{15}(n-7) + \frac{2}{15}\cos\frac{6\pi}{15}(n-7) + 0.8\cos\frac{8\pi}{15}(n-7)$$

$$n=0\ ;\ h(0) = \frac{1}{15} + \frac{2}{15}\cos\left(\frac{-14\pi}{15}\right) + \frac{2}{15}\cos\left(\frac{-28\pi}{15}\right) + \frac{2}{15}\cos\left(\frac{-42\pi}{15}\right) + 0.8\cos\left(\frac{-56\pi}{15}\right) = 0.4855$$

$$n=1\ ;\ h(1) = \frac{1}{15} + \frac{2}{15}\cos\left(\frac{-12\pi}{15}\right) + \frac{2}{15}\cos\left(\frac{-24\pi}{15}\right) + \frac{2}{15}\cos\left(\frac{-36\pi}{15}\right) + 0.8\cos\left(\frac{-48\pi}{15}\right) = -0.606$$

$$n=2\ ;\ h(2) = \frac{1}{15} + \frac{2}{15}\cos\left(\frac{-10\pi}{15}\right) + \frac{2}{15}\cos\left(\frac{-20\pi}{15}\right) + \frac{2}{15}\cos\left(\frac{-30\pi}{15}\right) + 0.8\cos\left(\frac{-40\pi}{15}\right) = -0.3333$$

$$n=3\ ;\ h(3) = \frac{1}{15} + \frac{2}{15}\cos\left(\frac{-8\pi}{15}\right) + \frac{2}{15}\cos\left(\frac{-16\pi}{15}\right) + \frac{2}{15}\cos\left(\frac{-24\pi}{15}\right) + 0.8\cos\left(\frac{-32\pi}{15}\right) = 0.6943$$

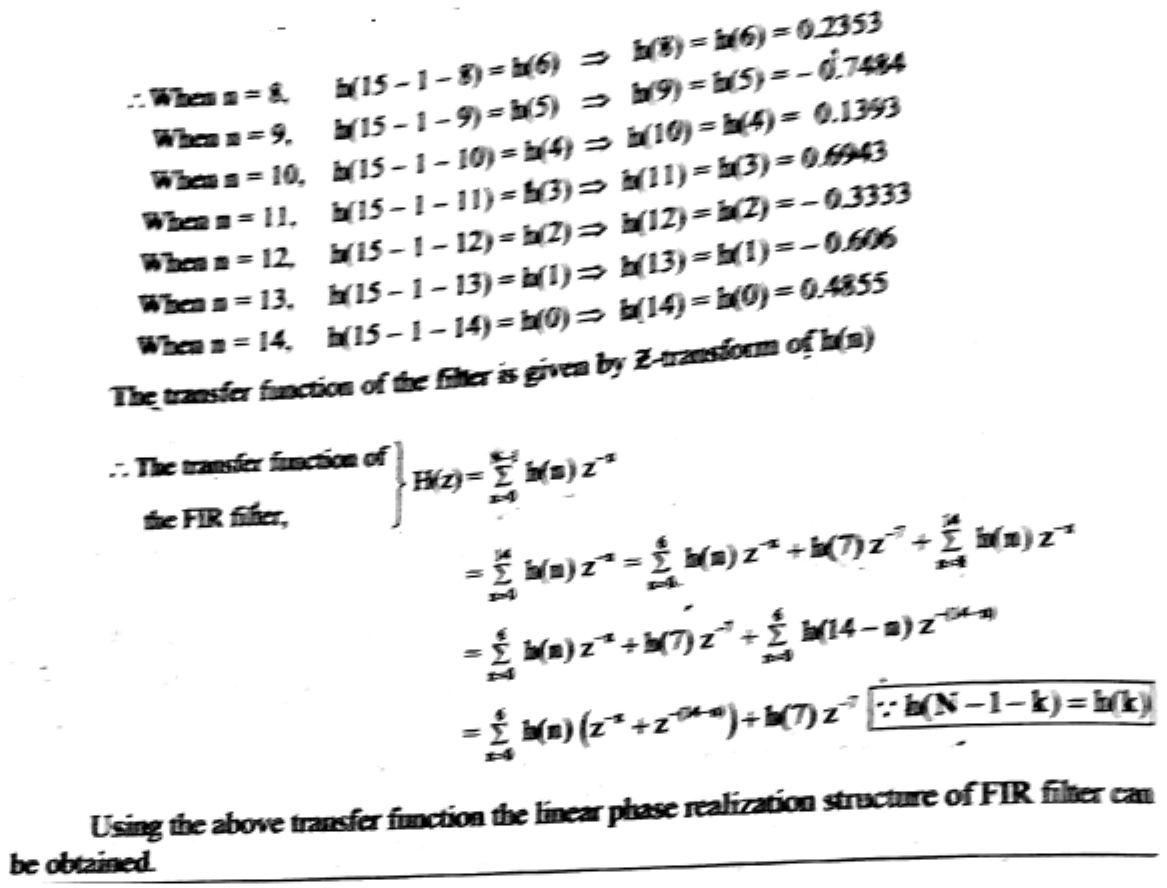
$$n=4\ ;\ h(4) = \frac{1}{15} + \frac{2}{15}\cos\left(\frac{-6\pi}{15}\right) + \frac{2}{15}\cos\left(\frac{-12\pi}{15}\right) + \frac{2}{15}\cos\left(\frac{-18\pi}{15}\right) + 0.8\cos\left(\frac{-24\pi}{15}\right) = 0.1393$$

$$n=5\ ;\ h(5) = \frac{1}{15} + \frac{2}{15}\cos\left(\frac{-4\pi}{15}\right) + \frac{2}{15}\cos\left(\frac{-8\pi}{15}\right) + \frac{2}{15}\cos\left(\frac{-12\pi}{15}\right) + 0.8\cos\left(\frac{-16\pi}{15}\right) = -0.7484$$

$$n=6\ ;\ h(6) = \frac{1}{15} + \frac{2}{15}\cos\left(\frac{-2\pi}{15}\right) + \frac{2}{15}\cos\left(\frac{-4\pi}{15}\right) + \frac{2}{15}\cos\left(\frac{-6\pi}{15}\right) + 0.8\cos\left(\frac{-8\pi}{15}\right) = 0.2353$$

$$n=7\ ;\ h(7) = \frac{1}{15} + \frac{2}{15}\cos 0 + \frac{2}{15}\cos 0 + \frac{2}{15}\cos 0 + 0.8\cos 0 = \frac{7}{15} + 0.8 = \boxed{1.2667}$$

For linear phase FIR filters the condition $h(N-1-n) = h(n)$ will be satisfied whe
$\alpha = N-1/2$.



$\therefore$ When $n = 8$, $h(15-1-8) = h(6) \Rightarrow h(8) = h(6) = 0.2353$

When $n = 9$, $h(15-1-9) = h(5) \Rightarrow h(9) = h(5) = -0.7484$

When $n = 10$, $h(15-1-10) = h(4) \Rightarrow h(10) = h(4) = 0.1393$

When $n = 11$, $h(15-1-11) = h(3) \Rightarrow h(11) = h(3) = 0.6943$

When $n = 12$, $h(15-1-12) = h(2) \Rightarrow h(12) = h(2) = -0.3333$

When $n = 13$, $h(15-1-13) = h(1) \Rightarrow h(13) = h(1) = -0.606$

When $n = 14$, $h(15-1-14) = h(0) \Rightarrow h(14) = h(0) = 0.4855$

The transfer function of the filter is given by Z-transform of $h(n)$

$\therefore$ The transfer function of the FIR filter,

$$H(z) = \sum_{n=0}^{N-1} h(n)\, z^{-n}$$

$$= \sum_{n=0}^{14} h(n)\, z^{-n} = \sum_{n=0}^{6} h(n)\, z^{-n} + h(7)\, z^{-7} + \sum_{n=8}^{14} h(n)\, z^{-n}$$

$$= \sum_{n=0}^{6} h(n)\, z^{-n} + h(7)\, z^{-7} + \sum_{n=0}^{6} h(14-n)\, z^{-(14-n)}$$

$$= \sum_{n=0}^{6} h(n)\left(z^{-n} + z^{-(14-n)}\right) + h(7)\, z^{-7} \quad \boxed{\because h(N-1-k) = h(k)}$$

Using the above transfer function the linear phase realization structure of FIR filter can be obtained.

3..5 Design of Optimum Equiripple Linear-Phase FIR

The window method and the frequency-sam pling method are relatively sim ple

techniques for designing linear-phase FIR filters. However, they also possess some minor disadvantages, , which may render them undesirable for some applications. A major problem is the lack of precise control of the critical frequencies such *ws.* The filter design method described in this section is formulated as a Chebyshev approximation problem . It is viewed as an optimum design criterion in the sense that the weighted approximation error between the desired frequency response and the actual frequency response is spread evenly across the passband and evenly across the stopband of the filter minimizing the maximum error. The resulting filter designs have ripples in both the passband and the stopband. To describe the design procedure, let us consider the design of a lowpass filter with passband edge frequency *a>p* and stopband edge frequency .

3..6 Structure realization of FIR Filters

In signal processing, a **digital filter** is a system that performs mathematical operations on a sampled, discrete-time signal to reduce or enhance certain aspects of that signal. This is in contrast to the other major type of electronic filter, the analog filter, which is anelectronic circuit operating on continuous-time analog signals.

A digital filter system usually consists of an analog-to-digital converter to sample the input signal, followed by a microprocessor and some peripheral components such as memory to store data and filter coefficients etc. Finally a digital-to-analog converter to complete the output stage. Program Instructions (software) running on the microprocessor implement the digital filter by performing the necessary mathematical operations on the numbers received from the ADC. In some high performance applications, an FPGA orASIC is used instead of a general purpose microprocessor, or a specialized DSP with specific paralleled architecture for expediting operations such as filtering.

Digital filters may be more expensive than an equivalent analog filter due to their increased complexity, but they make practical many designs that are impractical or impossible as analog filters. When used in the context of real-time analog systems, digital filters sometimes have problematic latency (the difference in time between the input and the response) due to the associated analog-to-digital and digital-to-analog conversions and anti-aliasing filters, or due to other delays in their implementation.

Digital filters are commonplace and an essential element of everyday electronics such as radios, cellphones, and AV receivers.

3.6.1. Characterization

A digital filter is characterized by its transfer function, or equivalently, its difference equation. Mathematical analysis of the transfer function can describe how it will respond to any input. As such, designing a filter consists of developing specifications appropriate to the problem (for example, a second-order low pass filter with a specific cut-off frequency), and then producing a transfer function which meets the specifications.

The transfer function for a linear, time-invariant, digital filter can be expressed as a transfer function in the *Z*-domain; if it is causal, then it has the form:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M}}$$

where the order of the filter is the greater of *N* or *M*. See *Z*-transform's LCCD equation for further discussion of this transfer function.

This is the form for a recursive filter with both the inputs (Numerator) and outputs (Denominator), which typically leads to an IIR infinite impulse response behaviour, but if thedenominator is made equal to unity i.e. no feedback, then this becomes an FIR or finite impulse response filter.

The impulse response, often denoted $h(k)$ or h_k, is a measurement of how a filter will respond to the Kronecker delta function. Digital filters are typically considered in two categories: infinite impulse response (IIR) and finite impulse response (FIR). In the case of linear time-invariant FIR filters, the impulse response is exactly equal to the sequence of filter coefficients:

$$y_n = \sum_{k=0}^{n-1} h_k x_{n-k}$$

IIR filters on the other hand are recursive, with the output depending on both current and previous inputs as well as previous outputs. The general form of an IIR filter is thus:

$$\sum_{m=0}^{M-1} a_m y_{n-m} = \sum_{k=0}^{n-1} b_k x_{n-k}$$

Plotting the impulse response will reveal how a filter will respond to a sudden, momentary disturbance.

1.Difference equation

In discrete-time systems, the digital filter is often implemented by converting the transfer function to a linear constant-coefficient difference equation (LCCD) via the Z-transform. The discrete frequency-domain transfer function is written as the ratio of two polynomials. For example:

$$H(z) = \frac{(z+1)^2}{(z-\frac{1}{2})(z+\frac{3}{4})}$$

This is expanded:

$$H(z) = \frac{z^2 + 2z + 1}{z^2 + \frac{1}{4}z - \frac{3}{8}}$$

and to make the corresponding filter causal, the numerator and denominator are divided by the highest order of z:

$$H(z) = \frac{1 + 2z^{-1} + z^{-2}}{1 + \frac{1}{4}z^{-1} - \frac{3}{8}z^{-2}} = \frac{Y(z)}{X(z)}$$

The coefficients of the denominator, a_k, are the 'feed-backward' coefficients and the coefficients of the numerator are the 'feed-forward' coefficients, b_k . The resultant linear difference equation is:

$$y[n] = -\sum_{k=1}^{M} a_k y[n-k] + \sum_{k=0}^{N} b_k x[n-k]$$

or, for the example above:

$$\frac{Y(z)}{X(z)} = \frac{1 + 2z^{-1} + z^{-2}}{1 + \frac{1}{4}z^{-1} - \frac{3}{8}z^{-2}}$$

rearranging terms:

$$\Rightarrow (1 + \frac{1}{4}z^{-1} - \frac{3}{8}z^{-2})Y(z) = (1 + 2z^{-1} + z^{-2})X(z)$$

then by taking the inverse *z*-transform:

$$\Rightarrow y[n] + \frac{1}{4}y[n-1] - \frac{3}{8}y[n-2] = x[n] + 2x[n-1] + x[n-2]$$

and finally, by solving for $y[n]$:

$$y[n] = -\frac{1}{4}y[n-1] + \frac{3}{8}y[n-2] + x[n] + 2x[n-1] + x[n-2]$$

This equation shows how to compute the next output sample, $y[n]$, in terms of the past outputs, $y[n-p]$, the present input, $x[n]$, and the past inputs, $x[n-p]$. Applying the filter to an input in this form is equivalent to a Direct Form I or II realization, depending on the exact order of evaluationAfter a filter is designed, it must be *realized* by developing a signal flow diagram that describes the filter in terms of operations on sample sequences.

A given transfer function may be realized in many ways. Consider how a simple expression such as $ax + bx + c$ could be evaluated – one could also compute the equivalent $x(a + b) + c$. In the same way, all realizations may be seen as "factorizations" of the same transfer function, but different realizations will have different numerical properties. Specifically, some realizations are more efficient in terms of the number of operations or storage elements required for their implementation, and others provide advantages such as improved numerical stability and reduced round-off error. Some structures are better for fixed-point arithmetic and others may be better for floating-point arithmetic.

1.Direct Form I

A straightforward approach for IIR filter realization is Direct Form I, where the difference equation is evaluated directly. This form is practical for small filters, but may be inefficient and impractical (numerically unstable) for complex designs.[3] In general, this form requires 2N delay elements (for both input and output signals) for a filter of order N.

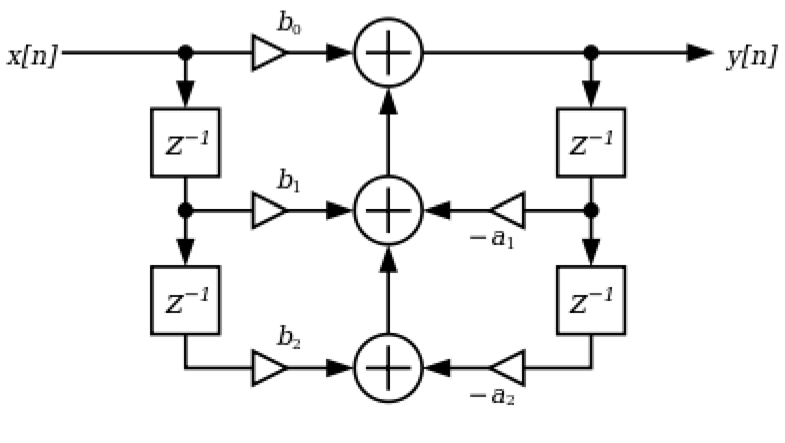


Fig.3.3. Direct form I

2.Direct Form II

The alternate Direct Form II only needs *N* delay units, where *N* is the order of the filter – potentially half as much as Direct Form I. This structure is obtained by reversing the order of the numerator and denominator sections of Direct Form I, since they are in fact two linear systems, and the commutativity property applies. Then, one will notice that there are two columns of delays (z^{-1}) that tap off the center net, and these can be combined since they are redundant, yielding the implementation as shown below.

The disadvantage is that Direct Form II increases the possibility of arithmetic overflow for filters of high Q or resonance.[4] It has been shown that as Q increases, the round-off noise of both direct form topologies increases without bounds.[5] This is because, conceptually, the signal is first passed through an all-pole filter (which normally boosts gain at the resonant frequencies) before the result of that is saturated, then passed through an all-zero filter (which often attenuates much of what the all-pole half amplifies).

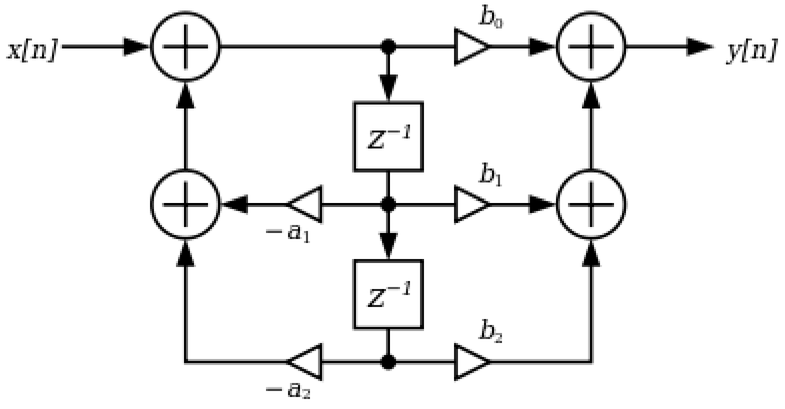


Fig.3.4. Direct form II

3.Cascaded second-order sections

A common strategy is to realize a higher-order (greater than 2) digital filter as a cascaded series of second-order "biquadratric" (or "biquad") sections[6] (see digital biquad filter). The advantage of this strategy is that the coefficient range is limited. Cascading direct form II sections results in N delay elements for filters of order N. Cascading direct form I sections results in N+2 delay elements since the delay elements of the input of any section (except the first section) are redundant with the delay elements of the output of the preceding section.

4.**Linear-Phase FIR Structures Phase FIR Structures**

The symmetry (or antisymmetry) property of a linear-phase FIR filter can be exploited to reduce the number of multipliers into almost half of that in the direct form implementations •

Consider a length-7 Type 1 FIR transfer function with a symmetric impulse response: $H(Z) = h(0) + h(1)Z^{-1} + h(2)Z^{-2} + h(3)Z^{-3} + h(2)Z^{-4} + h(1)Z^{-5} + h(0)Z^{-6}$.Rearranging, we get

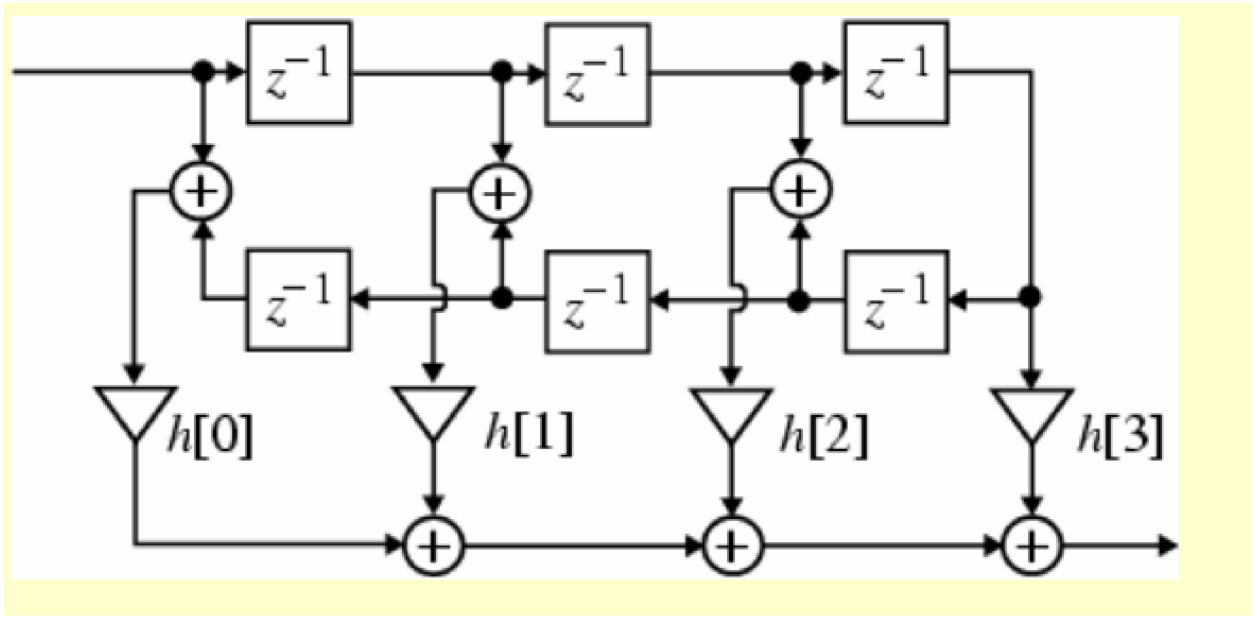


Fig.3.4. ***Linear phase FIRI***

5.Polyphase Polyphase FIR Structures FIR Structures

The polyphase decomposition of H(z) leads to a parallel form structure.

To illustrate this approach, consider a causal FIR transfer function H(z) with N = 8:

$$H(Z) = h(0) + h(1)Z^{-1} + h(2)Z^{-2} + h(3)Z^{-3} + h(4)Z^{-4} + h(5)Z^{-5} + h(6)Z^{-6} + h(7)Z^{-7} + h(8)Z^{-8}$$

H(z) can be expressed as a sum of two terms, with one term containing the even indexed coefficients and the other containing the odd-indexed coefficients:

$$H(Z) = h(0) + +h(2)Z^{-2} + h(4)Z^{-4} + h(6)Z^{-6} + h(8)Z^{-8} + Z^{-1}[h(1) + h(3)Z^{-2} + +h(5)Z^{-4} + h(7)Z^{-6}]$$

Putting $H(Z) = E_0(Z^2) + Z^{-1}E_1(Z^2)$.

The subfilters in the polyphase realization of an FIR transfer function are also FIR filters and can be realized using any methods. However, to obtain a canonic realization of the overall structure, the delays in all subfilters must be shared.

Part A

1. What is a high pass filter?
2. Compare analog and digital filters.
3. Name the techniques available for the design of analog filter.
4. Mention the requirement for a digital filter to be stable and causal.
5. What is frequency sampling method

Part B

6. Design a low pass digital filter of order 5 with cut of frequency 0.2 pi using hanning window.
7. An LTI system is described by y (n)+ y(n-1)- 0.25y(n-2)=x(n).Realize in direct form I and Cascade form.

SCHOOL OF ELECTRICAL AND ELECTRONICS

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

UNIT – IV - – FINITE WORD LENGTH EFFECTS -SEC1315

FINITE WORD LENGTH EFFECTS

4.1. Quantization Noise

Quantization refers to the process of approximating the continuous set of values in the data with a finite (preferably small) set of values. The input to a quantizer is the original data, and the output is always one among a finite number of levels. The quantizer is a function whose set of output values are discrete, and usually finite. Obviously, this is a process of approximation, and a good quantizer is one which represents the original signal with minimum loss or distortion. The difference between the actual analog value and quantized digital value due is called **quantization error**. This error is due either to rounding or truncation.

Because quantization is a many-to-few mapping, it is an inherently non-linear and irreversible process (i.e., because the same output value is shared by multiple input values, it is impossible in general to recover the exact input value when given only the output value).The set of possible input values may be infinitely large, and may possibly be continuous and therefore uncountable (such as the set of all real numbers, or all real numbers within some limited range). The set of possible output values may be finite or countably infinite. The input and output sets involved in quantization can be defined in a rather general way. For example, *vector quantization* is the application of quantization to multi-dimensional (vector-valued) input data.[1]

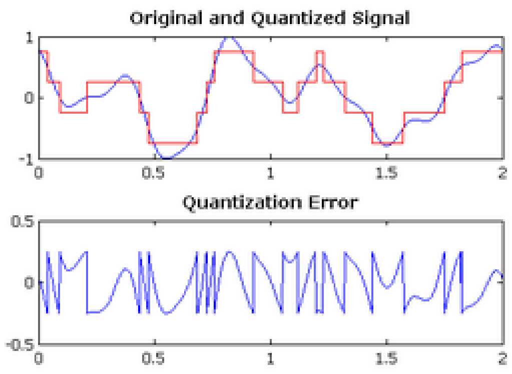


Fig. 4.1. ***Quantization process***

Quantization noise is a model of quantization error introduced by quantization in the analog-to-digital conversion (ADC) in telecommunication systems and signal processing. It is a rounding error between the analog input voltage to the ADC and the output digitized value. The noise is non-linear and signal-dependent. It can be modelled in several different ways. In an ideal analog-to-digital converter, where the quantization error is uniformly distributed between −1/2 LSB and +1/2 LSB, and the signal has a uniform distribution covering all quantization levels, the signal-to-noise ratio (SNR) can be calculated from

The most common test signals that fulfill this are full amplitude triangle waves and saw tooth waves. In this case a 16-bit ADC has a maximum signal-to-noise ratio of 6.0206 × 16 = 96.33 dB. When the input signal is a full-amplitude sine wave the distribution of the signal is no longer uniform, and the corresponding equation is instead Here, the quantization noise is once again assumed to be uniformly distributed. When the input signal has a high amplitude and a wide frequency spectrum this is the case. In this case a 16-bit ADC has a maximum signal-to-noise ratio of 98.09 dB. The 1.761 difference in signal-to-noise only occurs due to the signal being a full-scale sine wave instead of a triangle/saw tooth.

4.2. Derivation for Quantization Noise Power

Quantization noise is a model of quantization error introduced by quantization in the analog-to-digital conversion (ADC) intelecommunication systems and signal processing. It is a rounding error between the analog input voltage to the ADC and the output digitized value. The noise is non-linear and signal-dependent. It can be modelled in several different ways.

In an ideal analog-to-digital converter, where the quantization error is uniformly distributed between −1/2 LSB and +1/2 LSB, and the signal has a uniform distribution covering all quantization levels, the Signal-to-quantization-noise ratio (SQNR) can be calculated from $\mathrm{SQNR} = 20\log_{10}(2^Q) \approx 6.02 \cdot Q\ \mathrm{dB}$

Where Q is the number of quantization bits.The most common test signals that fulfill this are full amplitude triangle waves and sawtooth waves. For example, a 16-bit ADC has a maximum signal-to-noise ratio of 6.02 × 16 = 96.3 dB. When the input signal is a full-amplitude sine wave the distribution of the signal is no longer uniform, and the corresponding equation is instead $\mathrm{SQNR} \approx 1.761 + 6.02 \cdot Q\ \mathrm{dB}$

Here, the quantization noise is once again assumed to be uniformly distributed. When the input signal has a high amplitude and a wide frequency spectrum this is the case.[16] In this case a 16-

bit ADC has a maximum signal-to-noise ratio of 98.09 dB. The 1.761 difference in signal-to-noise only occurs due to the signal being a full-scale sine wave instead of a triangle/sawtooth. Quantization noise power can be derived from

$$N = \frac{(\delta v)^2}{12} W$$

where δv is the voltage of the level.

(Typical real-life values are worse than this theoretical minimum, due to the addition of dither to reduce the objectionable effects of quantization, and to imperfections of the ADC circuitry. Also see noise shaping.)

For complex signals in high-resolution ADCs this is an accurate model. For low-resolution ADCs, low-level signals in high-resolution ADCs, and for simple waveforms the quantization noise is not uniformly distributed, making this model inaccurate.[17] In these cases the quantization noise distribution is strongly affected by the exact amplitude of the signal.The calculations above, however, assume a completely filled input channel. If this is not the case - if the input signal is small - the relative quantization distortion can be very large. To circumvent this issue, analog compressors and expanders can be used, but these introduce large amounts of distortion as well, especially if the compressor does not match the expander. The application of such compressors and expanders is also known as companding.

4.3. Fixed Point And Floating Point Number Representations

4..1.1. Fixed Point Representation:

In computing, a **fixed-point number** representation is a real data type for a number that has a fixed number of digits after (and sometimes also before) the radix point (*e.g.*, after the decimal point '.' in English decimal notation). Fixed-point number representation can be compared to the more complicated (and more computationally demanding) floating point number representation. Fixed-point numbers are useful for representing fractional values, usually in base 2 or base 10, when the executing processor has no floating point unit (FPU) or if fixed-point provides improved performance or accuracy for the application at hand. Most low-cost embedded microprocessors and microcontrollers do not have an FPU.

The two most common fixed-point types are decimal and binary. Decimal fixed-point types have a scaling factor that is a power of ten, for binary fixed-point types it is a power of two. Binary fixed- Binary fixed-point types are most commonly used, because the rescaling operations can be implemented as fast bit shifts. Binary fixed-point numbers can represent fractional powers of two exactly, but, like binary floating-point numbers, cannot exactly represent fractional powers of ten. If exact fractional powers of ten are desired, then a decimal format should be used. For example, one-tenth (0.1) and one-hundredth (0.01) can be represented only approximately by binary fixed-point or binary floating-point representations, while they can be represented exactly in decimal fixed-point or decimal floating-point representations. The fixed point numbers consists of integers and fractions. The position of the radix point is fixed.

4..2.1. Floating Point Representation

In computing, **floating point** describes a system for numerical representation in which a string of digits (or bits) represents a rational number. The term *floating point* refers to the fact that the radix point (decimal point, or, more commonly in computers, binary point) can "float"; that is, it can be placed anywhere relative to the significant digits of the number. This position is indicated separately in the internal representation, and floating-point representation can thus be thought of as a computer realization of scientific notation. The advantage of floating-point representation over **fixed-point** (and integer) representation is that it can support a much wider range of values. For example, a fixed-point representation that has seven decimal digits, with the decimal point assumed to be positioned after the fifth digit, can represent the numbers 12345.67, 8765.43, 123.00, and so on, whereas a floating-point representation (such as the IEEE 754 decimal32 format) with seven decimal digits could in addition represent 1.234567, 123456.7, 0.00001234567, 1234567000000000, and so on. The floating-point format needs slightly more storage (to encode the position of the radix point), so when stored in the same space, floating-point numbers achieve their greater range at the expense of slightly less precision.

4..3.1. Comparison

Fixed Point Representation	*Floating Point Representation*
Decimal point is fixed	Decimal point is floating

Range is lesser	Wider than fixed point
Less precision	More precision
Less costly	More costly

4.4. Overflow error

*4..4.1. Truncation Error and Rounding Error***:**

A **round-off error**, also called **rounding error**, is the difference between the calculated approximation of a number and its exact mathematical value. Numerical analysis specifically tries to estimate this error when using approximation equations and/or algorithms, especially when using finite digits to represent real numbers (which in theory have infinite digits). This is a form of quantization error.

Truncation: simply chop off the remaining digits; also called rounding to zero. 0.142857 ≈ 0.142 (dropping all significant digits after 3rd) **Round to nearest**: round to the nearest value, with ties broken in one of two ways. The result may **round up** or **round down**. In mathematics, **truncation** is the term for limiting the number of digits right of the decimal point, by discarding the least significant ones. For example, consider the real numbers 5.6341432543653654 32.438191288 -6.3444444444444 To *truncate* these numbers to 4 decimal digits, we only consider the 4 digits to the right of the decimal point. The result would be: 5.6341 32.4381 -6.3444 Note that in some cases, truncating would yield the same result as rounding, but truncation does not round up or round down the digits; it merely cuts off at the specified digit. The truncation error can be twice the maximum error in rounding.\

4.5. . Coefficient Quantization Error

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.

Each filter structure has its own finite, generally non uniform grid so frealizable pole and zero locations when the filter coefficients are quantized to a finite word length. In general the pole and zero locations desired in filter do not correspond exactly to the realizable locations. The error in filter performance (usually measured in terms of a frequency response error) resulting from the placement of the poles and zeroes at the nonideal but realizable locations is referred to as coefficient quantization error.

4.6. Limit Cycle Oscillations

A limit cycle, sometimes referred to as a multiplier round off limit cycle , is a low-level oscillation that can exist in an otherwise stable filter as a result of the nonlinearity associated with rounding (or truncating) internal filter calculations Limit cycles require recursion to exist and do not occur in non recursive FIR filters Limit cycles are primarily of concern in fixed-point recursive filters. As long as floating-point filters are realized as the parallel or cascade connection of first- and second-order sub filters, limit cycles will generally not be a problem since limit cycles are practically not observable in first- and second-order systems implemented with 32-bit floating-point arithmetic There are at least three ways of dealing with limit cycles when fixed-point arithmetic is used. One is to determine a bound on the maximum limit cycle amplitude, expressed as an integral number of quantization steps   It is then possible to choose a word length that makes the limit cycle amplitude acceptably low. Alternately, limit cycles can be prevented by randomly rounding calculation supordown However, this approach is complicated to implement. The third approach is to properly choose the filter realization structure and then quantize the filter calculations using magnitude truncation]. This approach has the disadvantage of producing more round off noise than truncation or rounding   There are two types of limit cycles

(1) Granular limit cycle is usually of low amplitude

(2) Overflow limit cycle has large amplitudes

Two types of granular limit cycles have been observed in IIR digital filters:

(1) Inaccessible limit cycle - can appear only if the initial conditions of the digital filter at the time of starting pertain to that limit cycle

(2) Accessible limit cycle - can appear by starting the digital filter with initial conditions not pertaining to the limit cycle

T o illustrate the characteristics o f a lim it-cycle oscillation, let us consider a single-pole system described by the linear difference equation $y(n) = ay(n-1) + x(n)$

where the pole is at *z* = *a*. On the other hand, the actual system , which is described by the nonlinear difference equation $v(n) = Q[av(n-1)] + x(n)$.

Suppose that the actual system is implemented with fixed-point arithmetic based on four bits for the magnitude plus a sign bit. The quantization that takes place after multiplication is assumed to round the resulting product upward. In Table we list the response o f the actual system for four different locations of the p ole *z*= *a*, and an input x(n) = $\beta\delta(n)$ where β = 15/16, which has the binary representation 0.1111. Ideally, the response of the system should decay toward zero exponentially [i.e., *y(n)* = *a"* → 0 0 as *n*→ ∞]. In the actual system , however, the response *v(n)* reaches a steady-state periodic output sequence with a period that depends on the value o f the pole. When the pole is positive, the oscillations occur with a period N_p = 1, so that the output reaches a constant value of $\frac{1}{16}$ for $a = \frac{1}{2}$ and $\frac{1}{8}$ for $a = \frac{3}{4}$ On the other hand, when the pole is negative, the output sequence oscillates between positive and negative values ($\pm\frac{1}{16}$ for $a = -\frac{1}{2}$ and $\pm\frac{1}{8}$ for $= -\frac{3}{4}$) Hence the period is N_p = 2.These limit cycles occur as a result of the quantization effects in multiplications.

When the input sequence *x (n)* to the filter becomes zero, the output of the filter then, after a number of iterations, enters into the limit cycle. The output remains in the limit cycle until another input o f sufficient size is applied that drives the system out o f the limit cycle. Similarly, zero-input limit cycles occur from nonzero initial conditions with the input *x (n)* = 0 .The amplitudes of the output during a limit cycle are confined to a range o f values that is called the *dead band* of the filter.

n	$a = 0.1000$ $= \frac{1}{2}$		$a = 1.1000$ $= -\frac{1}{2}$		$a = 0.1100$ $= \frac{3}{4}$		$a = 1.1100$ $= -\frac{3}{4}$	
0	0.1111	$(\frac{15}{16})$	0.1111	$(\frac{15}{16})$	0.1011	$(\frac{11}{16})$	0.1011	$(\frac{11}{16})$
1	0.1000	$(\frac{8}{16})$	1.1000	$(-\frac{8}{16})$	0.1000	$(\frac{8}{16})$	1.1000	$(-\frac{8}{16})$
2	0.0100	$(\frac{4}{16})$	0.0100	$(\frac{4}{16})$	0.0110	$(\frac{6}{16})$	0.0110	$(\frac{6}{16})$
3	0.0010	$(\frac{2}{16})$	1.0010	$(-\frac{2}{16})$	0.0101	$(\frac{5}{16})$	1.0101	$(-\frac{5}{16})$
4	0.0001	$(\frac{1}{16})$	0.0001	$(\frac{1}{16})$	0.0100	$(\frac{4}{16})$	0.0100	$(\frac{4}{16})$
5	0.0001	$(\frac{1}{16})$	1.0001	$(-\frac{1}{16})$	0.0011	$(\frac{3}{16})$	1.0011	$(-\frac{3}{16})$
6	0.0001	$(\frac{1}{16})$	0.0001	$(\frac{1}{16})$	0.0010	$(\frac{2}{16})$	0.0010	$(\frac{2}{16})$
7	0.0001	$(\frac{1}{16})$	1.0001	$(-\frac{1}{16})$	0.0010	$(\frac{2}{16})$	1.0010	$(-\frac{2}{16})$
8	0.0001	$(\frac{1}{16})$	0.0001	$(\frac{1}{16})$	0.0010	$(\frac{2}{16})$	0.0010	$(\frac{2}{16})$

Table 1.Limit cycle oscillation

It is interesting to note that w hen the response o f the single-pole filter is in the limit cycle, the actual nonlinear system operates as an equivalent linear system with a pole at $z = 1$ w hen the pole is positive and $z = -1$ when the pole is negative.

4.6.1 Signal scaling

Saturation arithmetic as just described eliminates limit cycles du e to overflow , on the one hand, but on the other hand, it causes undesirable signal distortion due to the nonlinearity of the clipper. In order to limit the amount of nonlinear distortion, it is important to scale the input signal and the unit sample response, between the input and any internal summing node in the system , such that overflow becomes a rare event.

PART A

1. What is quantization noise.
2. Define limit cycle.
3. Compare rounding off and truncation.
4. What is signal scaling
5. Define truncation error.

PART B

6. Write a note on limit cycle oscillation.
7. What is quantization derive the quantization noise power.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL AND ELECTRONICS

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

UNIT – V - TMS320C2407 DSP CONTROLLER & PROGRAMMING FOR POWER ELECTRONICS APPLICATIONS -SEC1315

DIGITAL SIGNAL PROCESSOR

The TMS320C24x is a member of the TMS320 family of digital signal processors(DSPs). The ’C24x is designed to meet a wide range of digital motor control(DMC) and embedded control applications.

5.1.TMS320 Family Overview

The TMS320 family consists of fixed-point, floating-point, multiprocessor digital signal processors (DSPs), and fixed-point DSP controllers. TMS320 DSPs have an architecture designed specifically for real-time signal processing. The’C24x series of DSP controllers combines this real-time processing capability with controller peripherals to create an ideal solution for control system applications. The following characteristics make the TMS320 family the right choice for a wide range of processing applications:

1. Very flexible instruction set

2.Inherent operational flexibility

3.High-speed performance

4.Innovative parallel architecture

5.Cost effectiveness

5.2.TMS320C24x Series of DSP Controllers

Designers have recognized the opportunity to redesign existing DMC systems to use advanced algorithms that yield better performance and reduce system component count. DSPs enable:

1. Design of robust controllers for a new generation of inexpensive motors,such as AC induction, DC permanent magnet, and switched-reluctance motors.
2. Full variable-speed control of brushless motor types that have lower manufacturing cost and higher reliability.
3. Energy savings through variable-speed control, saving up to 25% of the energy used by fixed-speed controllers.

4. Increased fuel economy, improved performance, and elimination of hydraulic fluid in automotive electronic power steering (EPS) systems .

5. Reduced manufacturing and maintenance costs by eliminating hydraulic fluids in automotive electronic braking systems.

6. More efficient and quieter operation due to less generation of torque ripple, resulting in less loss of power, lower vibration, and longer life

7. Elimination or reduction of memory lookup tables through real-time polynomial calculation, thereby reducing system cost.

8. Use of advanced algorithms that can reduce the number of sensors required in a system.

9. Control of power switching inverters, along with control algorithm processing.

10. Single-processor control of multi motor systems

The ’C24x DSP controllers are designed to meet the needs of control-based applications. By integrating the high performance of a DSP core and the on-chip peripherals of a microcontroller into a single-chip solution, the ’C24x

series yields a device that is an affordable alternative to traditional microcontroller units (MCUs) and expensive multichip designs. At 20 million instructions per second (MIPS), the ’C24x DSP controllers offer significant performance over traditional 16-bit microcontrollers and microprocessors. Future derivatives of these devices will run at speeds higher than 20 MIPS. The 16-bit, fixed-point DSP core of the ’C24x device provides analog designers

a digital solution that does not sacrifice the precision and performance of their systems. In fact, system performance can be enhanced through the use of advanced control algorithms for techniques such as adaptive control,Kalman filtering, and state control. The ’C24x DSP controllers offer reliability and programmability. Analog control systems, on the other hand, are hardwired solutions and can experience performance degradation due to aging,component tolerance, and drift.The high-speed central processing unit (CPU) allows the digital designer to

process algorithms in real time rather than approximate results with look-up tables. When the instruction set of these DSP controllers (which incorporates both signal processing instructions and general-purpose control functions) is coupled with the extensive development support available for the ’C24x devices,it reduces development time and provides the same ease of use as traditional 8- and 16-bit microcontrollers. The instruction set also allows you to

retain your software investment when moving from other general-purpose TMS320 fixed-point DSPs. It is source- and object-code compatible with the other members of the ’C24x generation, source code compatible with the ’C2x

generation, and upwardly source code compatible with the ’C5x generation of DSPs from Texas Instruments. The ’C24x architecture is also well-suited for processing control signals. It uses a 16-bit word length along with 32-bit registers for storing intermediate results, and has two hardware shifters available to scale numbers independently

of the CPU. This combination minimizes quantization and truncation errors, and increases processing power for additional functions. Two examples of these additional functions are: a notch filter that cancels mechanical resonances in a system, and an estimation technique that eliminates state sensors in a system. The ’C24x DSP controllers take advantage of an existing set of peripheral functions that allow Texas Instruments to quickly configure various series members for different price/performance points or for application optimization.

This library of both digital and mixed-signal peripherals includes :

Timers

Serial communications ports (SCI, SPI)

Analog-to-digital converters (ADC)

Event manager

system protection, such as watchdog timers

5.3.Architectural Overview

The ’C24x DSP uses an advanced, modified Harvard architecture that maximizes processing power by maintaining separate bus structures for program memory and data memory.

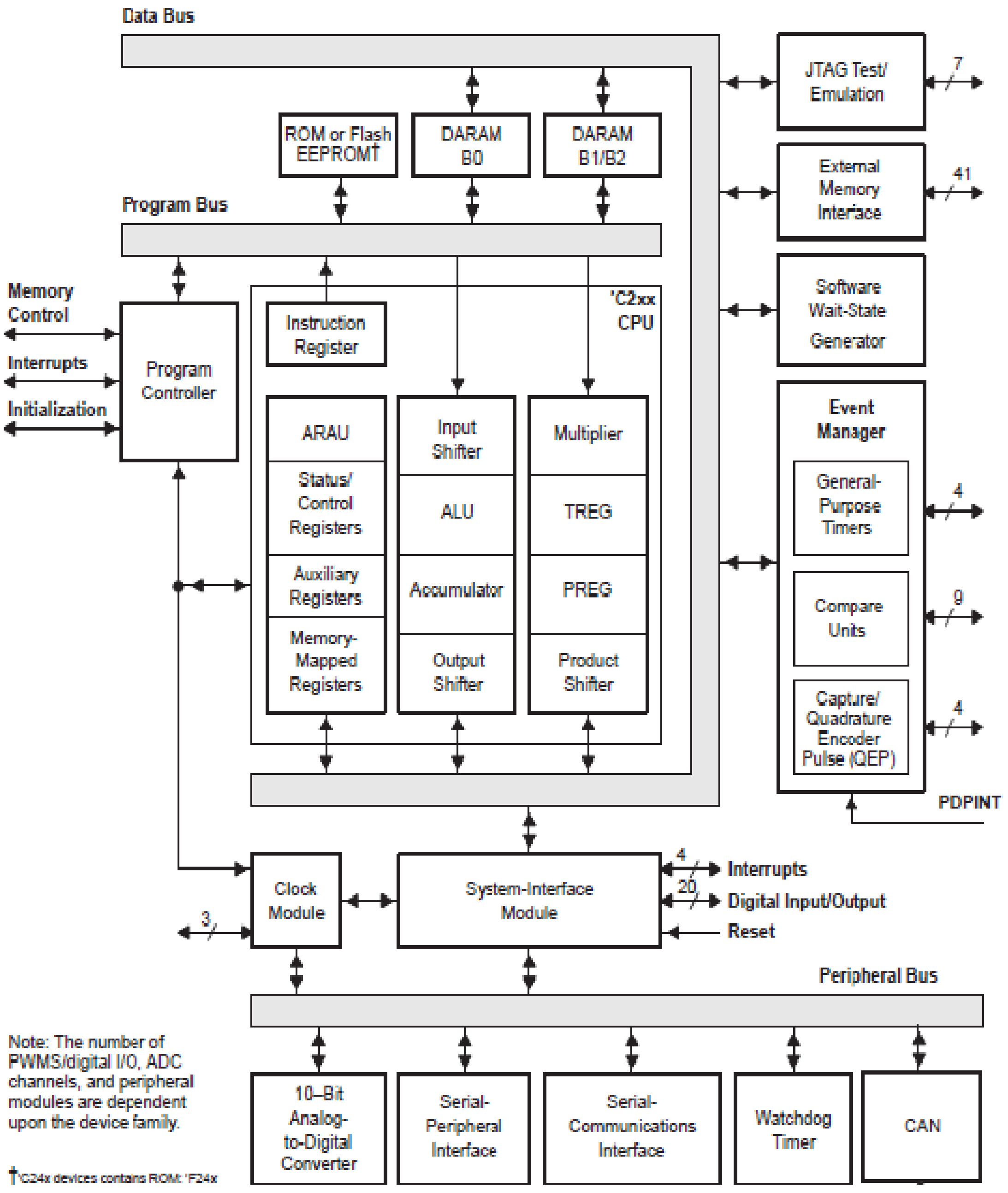


Fig.5.1. Architecture of TMS320C24X processor

5.3.1. C24x CPU Internal Bus Structure

The ’C24x DSP, a member of the TMS320 family of DSPs, includes a ’C2xx DSP core designed using the ’2xLP ASIC core. The ’C2xx DSP core has an internal data and program bus structure that is divided into six 16-bit buses. The six buses are:

PAB. The program address bus provides addresses for both reads from and writes to program emory.

DRAB. The data-read address bus provides addresses for reads from data memory.

DWAB. The data-write address bus provides addresses for writes to data memory.

PRDB. The program read bus carries instruction code and immediate operands, as well as table information, from program memory to the CPU.

DRDB. The data-read bus carries data from data memory to the central arithmetic logic unit (CALU) and the auxiliary register arithmetic unit (ARAU).

DWEB. The data-write bus carries data to both program memory and data memory. Having separate address buses for data reads (DRAB) and data writes (DWAB) allows the CPU to read and write in the same machine cycle.

5.3.2. Memory

The ’C24x contains the following types of on-chip memory:

Dual-access RAM (DARAM)

Flash EEPROM or ROM (masked)

The ’C24x memory is organized into four individually-selectable spaces:

Program (64K words)

Local data (64K words)

Global data (32K words)

Input/Output (64K words)

These spaces form an address range of 224K words.

1.On-Chip Dual-Access RAM (DARAM)

The ’C24x has 544 words of on-chip DARAM, which can be accessed twice per machine cycle. This memory is primarily intended to hold data, but when needed, can also be used to hold programs. The memory can be configured in one of two ways, depending on the state of the CNF bit in status register ST1.

When CNF = 0, all 544 words are configured as data memory.

When CNF = 1, 288 words are configured as data memory and 256 words are configured as program memory.

Because DARAM can be accessed twice per cycle, it improves the speed of the CPU. The CPU operates within a 4-cycle pipeline. In this pipeline, the CPU reads data on the third cycle and writes data on the fourth cycle. However, DARAM allows the CPU to write and read in one cycle; the CPU writes to DARAM on the master phase of the cycle and reads from DARAM on the slave phase. For example, suppose two instructions, A and B, store the accumulator value to DARAM and load the accumulator with a new value from DARAM. Instruction A stores the accumulator value during the master phase of the CPU cycle, and instruction B loads the new value in the accumulator during the slave phase. Because part of the dual-access operation is a write, it only applies to RAM.

2.Flash EEPROM

Flash EEPROM provides an attractive alternative to masked program ROM.Like ROM, flash is a nonvolatile memory type; however, it has the advantage of in-target reprogrammability. The ’F24x incorporates one 16K/8K □□16-bit flash EEPROM module in program space. This type of memory expands the capabilities of the ’F24x in the areas of prototyping, early field testing, and single-chip applications.Unlike most discrete flash memory, the ’F24x flash does not require a dedicated state machine because the algorithms for programming and erasing the flash are executed by the DSP core. This enables several advantages, including reduced chip size and sophisticated adaptive algorithms. For production programming, the IEEE Standard 1149.1 (JTAG) scan port provides easy access to on-chip RAM for downloading the algorithms and flash code. Other key features of the flash include zero-wait-state access rate and single 5-V power supply.An erased bit in the ’24x flash is read as a logic one, and a programmed bit is read as a logic zero. The flash requires a block-erase of the entire 16K/8K module; however, any combination of bits can be programmed. The following four algorithms are required for flash operations: clear, erase, flash-write, and program. For an explanation of these algorithms and a complete description of the flash EEPROM, see TMS320F20x/F24x DSPs Embedded Flash Memory Technical Reference (Literature number SPRU282).

3.Flash Serial Loader

Most of the on-chip flash devices are shipped with a serial bootloader code programmed at the following addresses: 0x0000 – 0x00FFh. All other flash addresses are in an erased state. The serial bootloader can be used to program the on-chip flash memory with user's code. During the flash programming sequence, the on-chip data RAM is used to load and execute the clear, erase, and program algorithms.

4.Factory-Masked ROM

For large-volume applications consisting of stable software free of bugs, lowcost, masked ROM is available and supported up to 16K or 4K words. If you want a custom ROM, you can provide the code or data to be programmed into the ROM in object-file format, and Texas Instruments will generate the appropriate process mask to program the ROM. For details, see Appendix B, Submitting ROM Codes to TI.A small portion of the ROM (128 or 64 words) is reserved by Texas Instruments for test purposes. These reserved locations are at addresses 0x3F80 or 3FC0 through 0x3FFF. This leaves about 16K words available for your code.

5.External Memory Interface Module

In addition to full, on-chip memory support, some of the ’C24x devices provide access to external memory by way of the External Memory Interface Module. This interface provides 16 external address lines, 16 external data lines, and relevant control signals to select data, program, and I/O spaces. An on-chip

wait-state generator allows interfacing with slower off-chip memory and peripherals.

5.3.3. Central Processing Unit

The ’C24x is based on TI's ’C2xx CPU. It contains:

A 32-bit central arithmetic logic unit (CALU)

A 32-bit accumulator

Input and output data-scaling shifters for the CALU

A 16-bit

16-bit multiplier

A product-scaling shifter

Data-address generation logic, which includes eight auxiliary registers and an auxiliary register arithmetic unit (ARAU)

Program-address generation logic.

5.3.4. Central Arithmetic Logic Unit (CALU) and Accumulator

The ’C24x performs 2s-complement arithmetic using the 32-bit CALU. The CALU uses 16-bit words taken from data memory, derived from an immediate instruction, or from the 32-bit multiplier result. In addition to arithmetic operations, the CALU can perform Boolean operations. The accumulator stores the output from the CALU; it can also provide a second input to the CALU. The accumulator is 32 bits wide and is divided into a highorder word (bits 31 through 16) and a low-order word (bits 15 through 0). Assembly language instructions are provided for storing the high- and loworder accumulator words to data memory.

1.Scaling Shifters

The ’C24x has three 32-bit shifters that allow for scaling, bit extraction, extended arithmetic, and overflow-prevention operations:

a**.Input data-scaling shifter (input shifter).** This shifter left-shifts 16-bit input data by 0 to 16 bits to align the data to the 32-bit input of the CALU.

b.**Output data-scaling shifter (output shifter).** This shifter left-shift output from the accumulator by 0 to 7 bits before the output is stored to data memory. The content of the accumulator remains unchanged.

c**.Product-scaling shifter (product shifter)**. The product register (PREG) receives the output of the multiplier. The product shifter shifts the output of the PREG before that output is sent to the input of the CALU. The product shifter has four product shift modes (no shift, left shift by one bit, left shift by four bits, and right shift by six bits), which are useful for performing multiply/accumulate operations, performing fractional arithmetic, or justifying fractional products.

5.3.5. Multiplier

The on-chip multiplier performs 16-bit □□16-bit 2s-complement multiplication with a 32-bit result. In conjunction with the multiplier, the ’C24x uses the 16-bit temporary register (TREG) and the 32-bit product register (PREG); TREG always supplies one of the values to be multiplied, and PREG receives the result

of each multiplication. Using the multiplier, TREG, and PREG, the ’C24x efficiently performs fundamental

DSP operations such as convolution, correlation, and filtering. The effective execution time of each multiplication instruction can be as short as one CPU cycle.

5.3.6. Auxiliary Register Arithmetic Unit (ARAU) and Auxiliary Registers

The ARAU generates data memory addresses when an instruction uses indirect addressing to access data memory.The ARAU is supported by eight auxiliary registers (AR0 through AR7), each of which can be loaded with a 16-bit value from data memory or directly from an instruction word. Each auxiliary register value can also be stored in data memory. The auxiliary registers are referenced by a 3-bit auxiliary register pointer (ARP) embedded in status register ST0.

5.3.7. Program Control

Several hardware and software mechanisms provide program control: Program control logic decodes instructions, manages the 4-level pipeline, stores the status of operations, and decodes conditional operations. Hardware elements included in the program control logic are the program counter, the status registers, the stack, and the address-generation logic. Software mechanisms used for program control include branches, calls, conditional instructions, a repeat instruction, reset, interrupts, and power down modes.

5.3.8. Serial-Scan Emulation

The ’C24x has seven pins dedicated to the serial scan emulation port (JTAG port). This port allows for non-intrusive emulation of ’C24x devices, and is supported by Texas Instruments emulation tools and by many third party debugger tools.

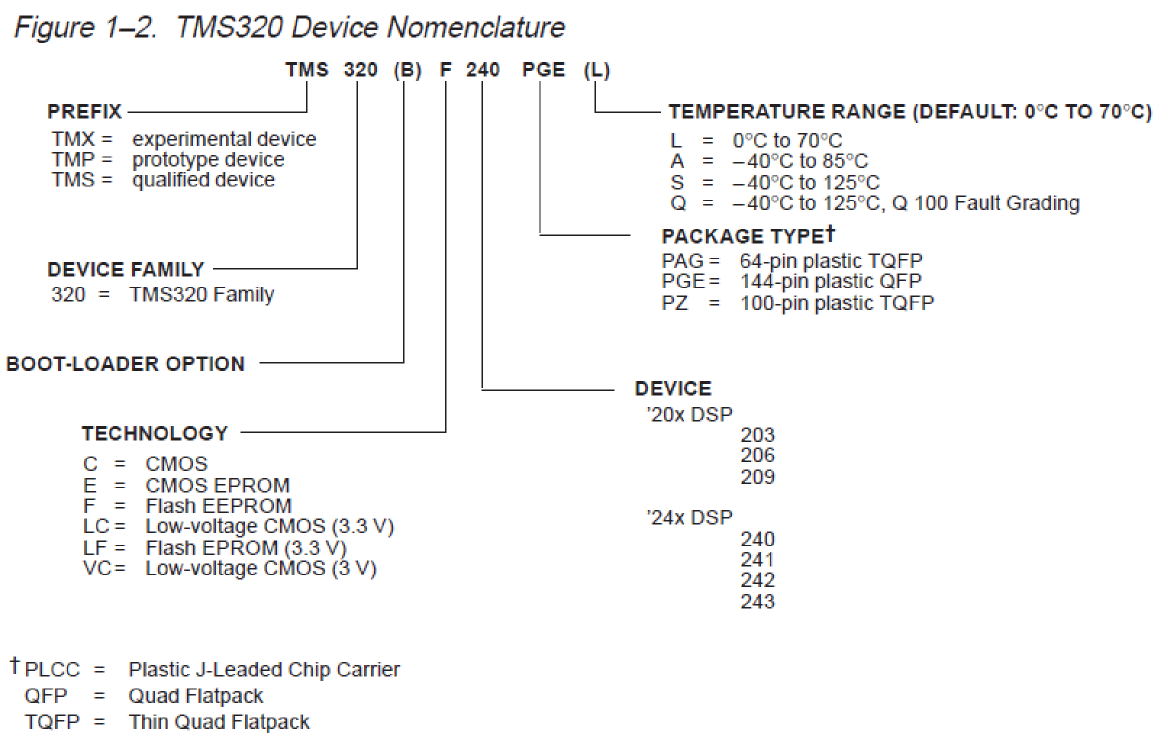


Figure 1–2. TMS320 Device Nomenclature
TMS 320 (B) F 240 PGE (L)
PREFIX
TMX = experimental device
TMP = prototype device
TMS = qualified device
DEVICE FAMILY
320 = TMS320 Family
BOOT-LOADER OPTION
TECHNOLOGY
C = CMOS
E = CMOS EPROM
F = Flash EEPROM
LC = Low-voltage CMOS (3.3 V)
LF = Flash EPROM (3.3 V)
VC = Low-voltage CMOS (3 V)
TEMPERATURE RANGE (DEFAULT: 0°C TO 70°C)
L = 0°C to 70°C
A = –40°C to 85°C
S = –40°C to 125°C
Q = –40°C to 125°C, Q 100 Fault Grading
PACKAGE TYPE†
PAG = 64-pin plastic TQFP
PGE = 144-pin plastic QFP
PZ = 100-pin plastic TQFP
DEVICE
'20x DSP
203
206
209
'24x DSP
240
241
242
243
† PLCC = Plastic J-Leaded Chip Carrier
QFP = Quad Flatpack
TQFP = Thin Quad Flatpack

Fig.5.2. TMS320C24X processor nomenclature

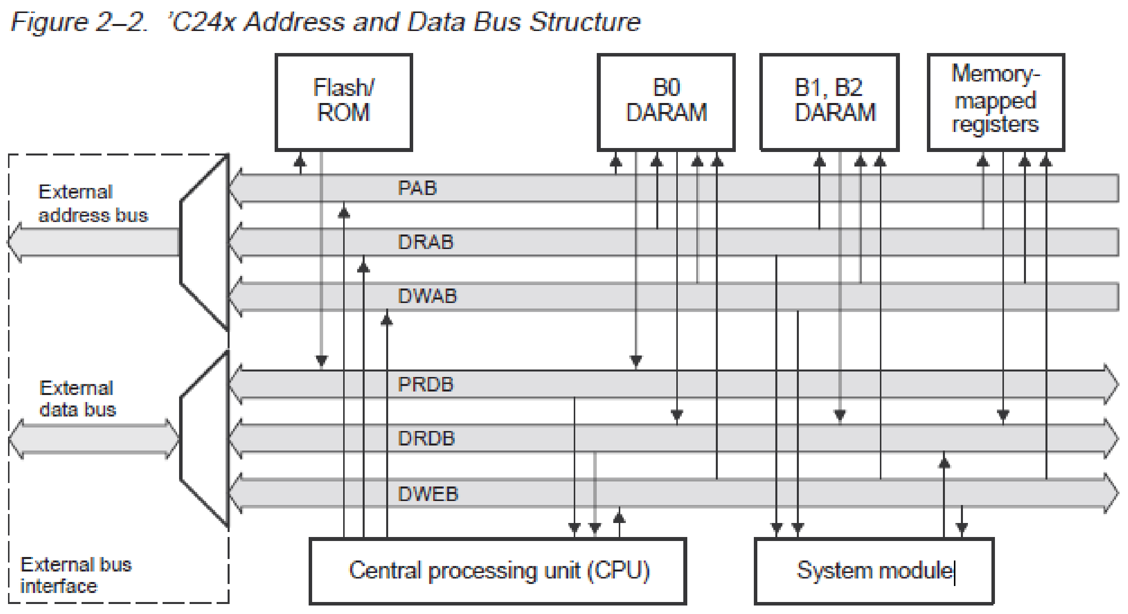


Fig.5.3. Bus structure

5.4. Memory and I/O Spaces

The ’C24x has a 16-bit address line that accesses four individually selectable spaces (224K words total):

A 64K-word program space

A 64K-word local data space

A 32K-word global data space

A 64K-word I/O space

5.4.1. Overview of Memory and I/O Spaces

The ’C24x design is based on an enhanced Harvard architecture. The ’C24x has multiple memory spaces accessible on three parallel buses: a program address bus (PAB), a data-read address bus (DRAB), and a data-write address bus (DWAB). Each of the three buses access different memory spaces for different phases of the device’s operation. Because the bus operations are independent, it is possible to access both the program and data spaces simultaneously. Within a given machine cycle, the CALU can execute as many as three concurrent memory operations. The ’C24x address map is organized into four individually selectable spaces:

1.Program memory (64K words) contains the instructions to be executed, as well as data used during program execution.

2.Data memory (64K words) holds data used by the instructions.

3.Global data memory (32K words) shares data with other devices or serves as additional data space.

4.Input/output (I/O) space (64K words) interfaces to external peripherals and may contain on-chip registers.

These spaces provide a total address space of 224K words. The ’C24x includes on-chip memory to aid in system performance and integration, and numerous addresses that can be used for external memory and I/O devices. The advantages of operating from on-chip memory are:

Higher performance than external memory (because the wait states required for slower external memories are avoided)

Lower cost than external memory,Lower power consumption than external memory

The advantage of operating from external memory is the ability to access a larger address space.The memory maps are generic for all ’C24x devices; however, each device has its own set of memory maps. ’C24x devices are available with different combinations of on-chip memory and peripherals.

5.4.2. Program Memory

The program-memory space is where the application program code resides; it can also hold table information and immediate operands. The program memory space addresses up to 64K 16-bit words. On the ’C24x device, these words include on-chip DARAM and on-chip ROM/flash EEPROM. When the ’C24x generates an address outside the set of addresses configured to on chip program memory, the device automatically generates an external access, asserting the appropriate control signals (if an external memory interface is present).

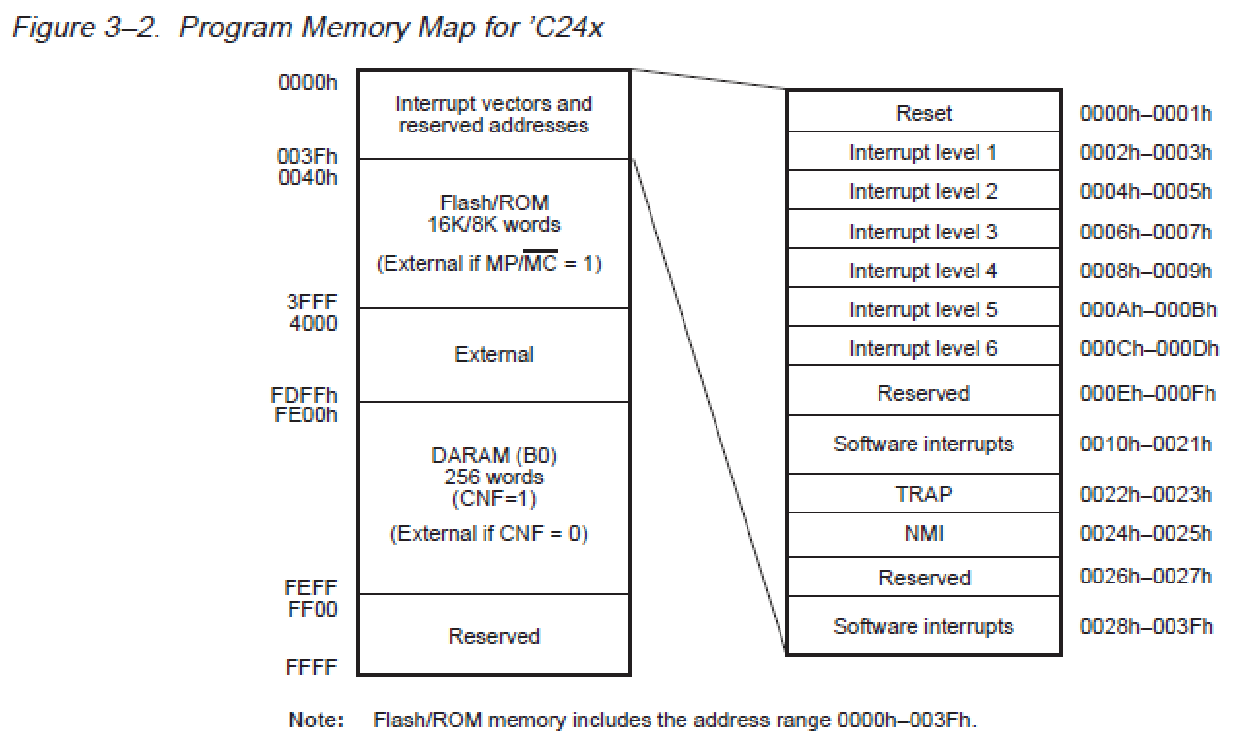


Fig.5.4. Program memory

5.4.3. Program Memory Configuration

Depending on which types of memory are included in a particular ’C24x device, two factors contribute to the configuration of program memory:

CNF bit. The CNF bit (bit 12) of status register ST1 determines whether the addresses for DARAM B0 are available for program space:

CNF = 0. There is no addressable on-chip program DARAM.

CNF = 1. The 256 words of DARAM B0 are configured for program use. At reset, any words of program/data DARAM are mapped into local data space (CNF = 0).

MP/MC pin. The level on the MP/MC pin determines whether program instructions are read from on-chip ROM or flash EEPROM (if available) after reset:

MP/MC = 0. The device is configured as a microcomputer. The onchip ROM/flash EEPROM is accessible. The device fetches the reset vector from on-chip memory.

MP/MC = 1. The device is configured as a microprocessor. The device fetches the reset vector from external memory. Regardless of the value of MP/MC, the ’C24x fetches its reset vector at location 0000h of program memory.

5.4.5. Data Memory

Data-memory space addresses up to 64K 16-bit words. Each ’C24x device has three on-chip DARAM blocks: B0, B1, and B2. Block B0 is configurable as either data memory or program memory. Blocks B1 and B2 are available for data memory only. Data memory can be addressed with either of two addressing modes: directaddressing or indirect-addressing. When direct addressing is used, data memory is addressed in blocks of 128 words called data pages. The entire 64K of data memory consists of 512 data

pages labeled 0 through 511. The current data page is determined by the value in the 9-bit data page pointer (DP) in status register ST0. Each of the 128 words on the current page is referenced by a 7-bit offset, which is taken from the instruction that is using direct addressing. Therefore, when an instruction uses direct addressing, you must specify both the data page (with a preceding instruction) and the offset (in the instruction that accesses data memory).

1.Data Page 0 Address Map

The data memory also includes the device’s memory-mapped registers (MMR), which reside at the top of data page 0 (addresses 0000h–007Fh). The three registers that can be accessed with zero wait states are Interrupt mask register (IMR),Global memory allocation register (GREG),Interrupt flag register (IFR),The test/emulation reserved area is used by the test and emulation systems for s pecial information transfers. The scratch-pad RAM block (B2) includes 32 words of DARAM that provide for variable storage without fragmenting the larger RAM blocks,whether internal or external. This RAM block supports dual-access operationsand can be addressed via any data-memory addressing mode.

Figure 3–3. Pages of Data Memory

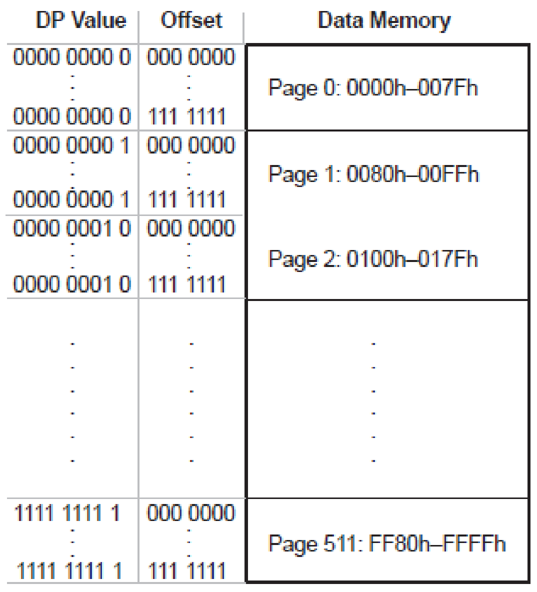


Fig.5.5. Pages of data memory

5.4.5. Data Memory Configuration

The following contributes to the configuration of data memory:

CNF bit. The CNF bit (bit 12) of status register ST1 determines whether the on-chip DARAM B0 is mapped into local data space or into program space.

CNF = 1. DARAM B0 is used for program space._ **CNF = 0.** B0 is used for data space.At reset, B0 is mapped into local data space (CNF = 0).

1.Global Data Memory

Addresses in the upper 32K words (8000h–FFFFh) of local data memory can be used for global data memory. The global memory allocation register (GREG) determines the size of the global data-memory space, which is between 256 and 32K words. The GREG is connected to the eight LSBs of the internal data bus and is memory-mapped to data-memory location 0005h. Table 3–2 shows the allowable GREG values and shows the corresponding address range set aside for global data memory. Any remaining addresses within 8000h–FFFFh are available for local data memory.

2.I/O Space

The I/O space memory addresses up to 64K 16-bit words. The I/O space is useful for mapping external peripherals and flash control registers. This I/O space is a generic space available for the ’C24x core. Depending on the specific device within the ’C24x family, the I/O space is partially available or disabled. External I/O space is available only in ’24x devices that have an external memory interface; otherwise, this space is reserved

5.5.Central Processing Unit

Figure 4–1. Block Diagram of the Input Scaling, Central Arithmetic Logic, and Multiplication Sections of the CPU

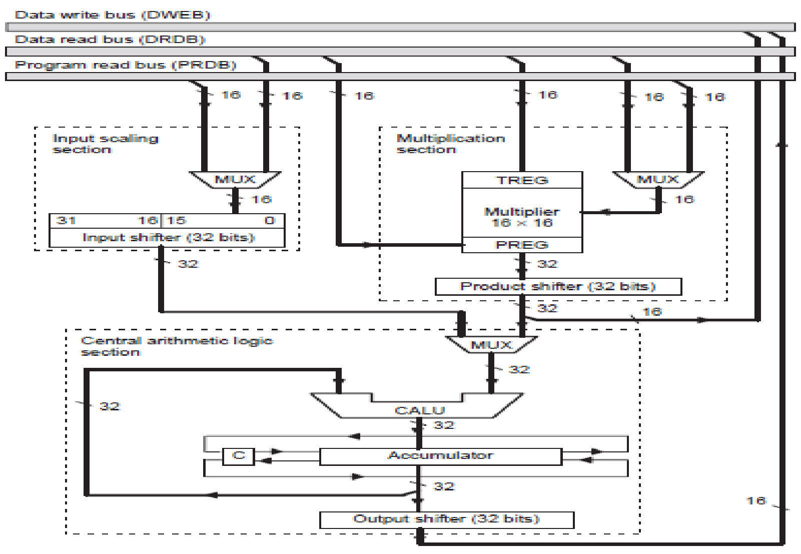


Fig.5.6. Central processing unit

5.4.1. Input Scaling Section

A 32-bit input data-scaling shifter (input shifter) aligns the 16-bit value from memory to the 32-bit central arithmetic logic unit (CALU). This data alignment is necessary for data-scaling arithmetic, as well as aligning masks for logical operations. The input shifter operates as part of the data path between program or data space and the CALU; and therefore, requires no cycle overhead. Described below are the input, output, and shift count of the input shifter.

Figure 4–2. Block Diagram of the Input Scaling Section

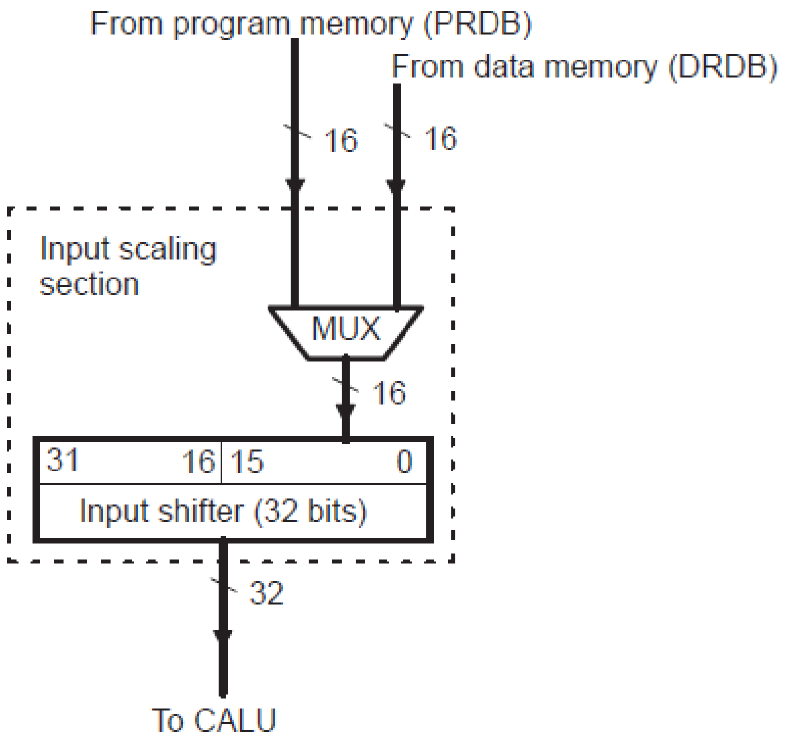


Fig.5.7. Input scaling unit

Figure 4.7, Block Diagram of the Input Scaling Section, can be used as a reference throughout the discussion.

1. Input. Bits 15 through 0 of the input shifter accept a 16-bit input from either of two sources (see Figure 4–2): The data read bus (DRDB). This input is a value from a data memory location referenced in an instruction operand. The program read bus (PRDB). This input is a constant value given as an instruction operand.

2.Output. After a value has been accepted into bits 15 through 0, the input shifter aligns the16-bit value to the 32-bit bus of the CALU as shown in Figure 4–2. The shifter shifts the value left 0 to 16 bits and then sends the 32-bit result to the CALU. During the left shift, unused LSBs in the shifter are filled with 0s, and unused MSBs in the shifter are either filled with 0s or sign extended, depending on the value of the sign-extension mode bit (SXM) of status register ST1.

3.Shift count. The shifter can left shift a 16-bit value by 0 to 16 bits. The size of the shift (or the shift count) is obtained from one of two sources: □ A constant embedded in the instruction word. Putting the shift count in the instruction word allows you to use specific data-scaling or alignment operations customized for your program code. □ The four LSBs of the temporary register (TREG). The TREG-based shift allows the data-scaling factor to be determined dynamically so that it can be adapted to the system's performance.

4.Sign-extension mode bit. For many (but not all) instructions, the sign-extension mode bit (SXM), bit 10 of status register ST1, determines whether the CALU uses sign extension during its calculations. If SXM = 0, sign extension is suppressed. If SXM = 1, the output of the input shifter is sign extended. Figure 4–3 shows an example of an input value shifted left by eight bits for SXM = 0. The MSBs of the value passed to the CALU are zero filled. Figure 4–4 shows the same shift but with SXM = 1. The value is sign extended during the shift.

5.4.2. Multiplication Section

The 'C24x uses a 16-bit □□16-bit hardware multiplier that can produce a signed or unsigned 32-bit product in a single machine cycle. As shown in Figure 4–5, the multiplication section consists of: □ The 16-bit temporary register (TREG), which holds one of the multiplicands □ The multiplier, which multiplies the TREG value by a second value from data memory or program memory □ The 32-bit product register (PREG), which receives the result of the multiplication □ The product shifter, which scales the PREG value before passing it to the CALU

1.Multiplier

The 16-bit □□16-bit hardware multiplier can produce a signed or unsigned 32-bit product in a single machine cycle. The two numbers being multiplied are treated as 2s-complement numbers, except during unsigned multiplication (MPYU instruction). Descriptions of the inputs to, and output of, the multiplier follow.

a.Inputs. The multiplier accepts two 16-bit inputs:

One input is always from the 16-bit temporary register (TREG). The TREG is loaded before the multiplication with a data-value from the data read bus (DRDB).The other input is one of the following:

_ A data-memory value from the data read bus (DRDB)

_ A program memory value from the program read bus (PRDB)

b.Output. After the two 16-bit inputs are multiplied, the 32-bit result is stored in the product register (PREG). The output of the PREG is connected to the 32-bit product-scaling shifter. Through this shifter, the product is transferred from the PREG to the CALU or to data memory (by the SPH and SPL instructions).

Figure 4–5. Block Diagram of the Multiplication Section

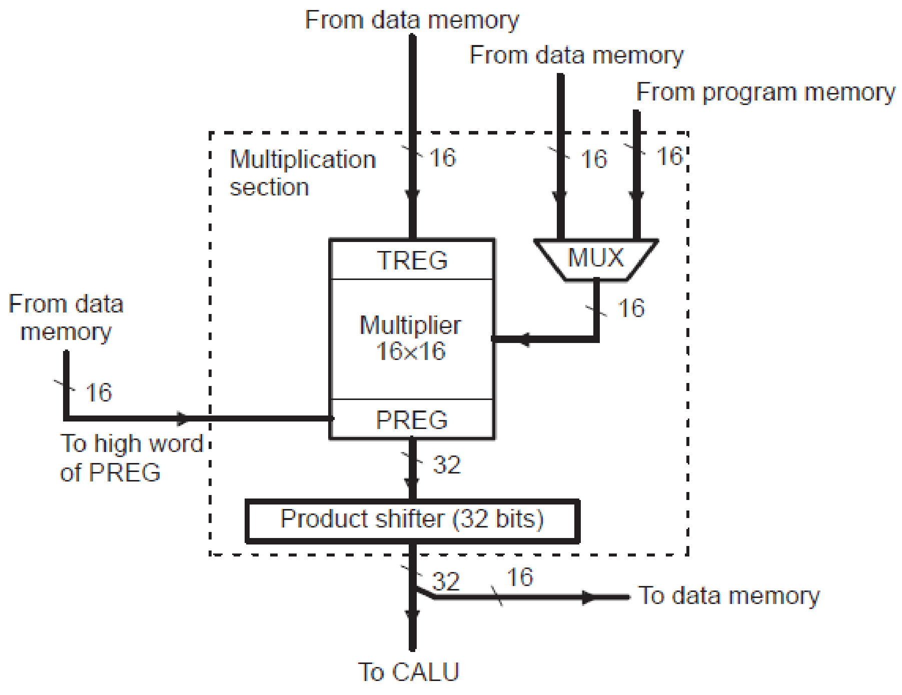


Fig.5.8. Multiplier- block diagram

2.Product-Scaling Shifter

The product-scaling shifter (product shifter) facilitates scaling of the product register (PREG) value. The shifter has a 32-bit input connected to the output of the PREG and a 32-bit output connected to the input of the CALU.

a.Input. The shifter has a 32-bit input connected to the output of the PREG.

b.Output. After the shifter completes the shift, all 32 bits of the result can be passed to the CALU, or 16 bits of the result can be stored to data memory.

c.Shift Modes. This shifter uses one of four product shift modes, summarized in Table 4–1. As shown in the table, these modes are determined by the product shift mode (PM) bits of status register ST1. In the first shift mode (PM = 00), the shifter does not shift the product at all before giving it to the CALU or to data memory. The next two modes cause left shifts (of one or four), which are useful for implementing fractional arithmetic or justifying products. The right-shift mode shifts the product by six bits, enabling the execution of up to 128 consecutive multiply-and-accumulate operations without causing the accumulator to overflow. Note that the content of the PREG remains unchanged; the value is copied to the product shifter and shifted there.

5.4.3. Central Arithmetic Logic Section

The main components of the central arithmetic logic section are:

1.The central arithmetic logic unit (CALU), which implements a wide range of arithmetic and logic functions

2.The 32-bit accumulator (ACC), which receives the output of the CALU and is capable of performing bit shifts on its contents with the help of the carry bit (C). Figure 4–6 shows the accumulator’s high word (ACCH) and low word (ACCL).

3.The output shifter, which can shift a copy of either the high word or low word of the accumulator before sending it to data memory for storage

Figure 4–6. Block Diagram of the Central Arithmetic Logic Section

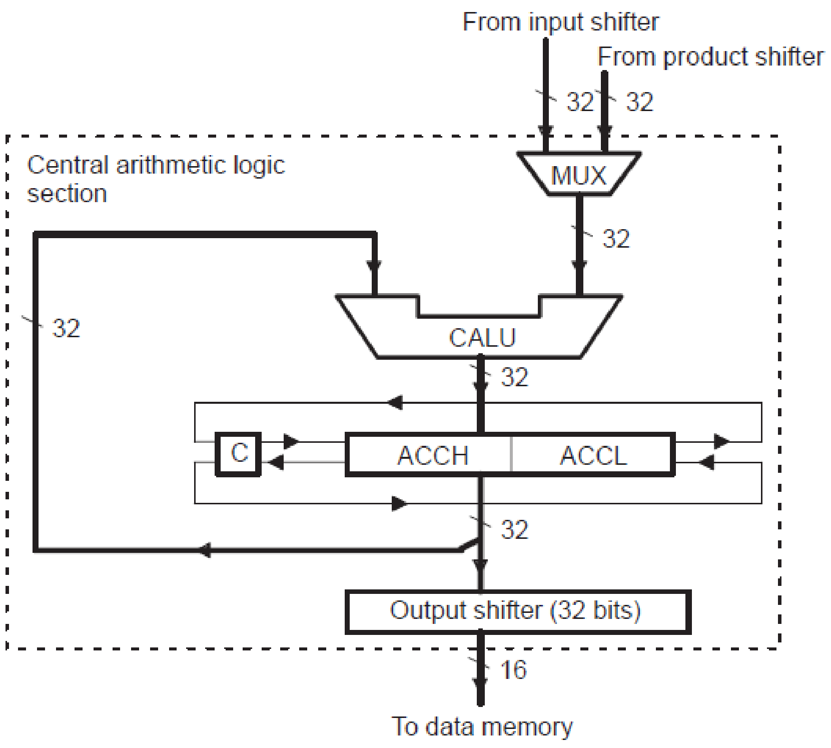


Fig.5.8. Block diagram -Central Arithmetic Logic Unit

1.Central Arithmetic Logic Unit (CALU)

The CALU implements a wide range of arithmetic and logic functions, most of which execute in a single clock cycle. These functions can be grouped into four categories:

16-bit addition,16-bit subtraction,Boolean logic operations,Bit testing, shifting, and rotating.Because the CALU can perform Boolean operations, you can perform bit manipulation. For bit shifting and rotating, the CALU uses the accumulator. The CALU is referred to as central because there is an independent arithmetic unit, the auxiliary register arithmetic unit (ARAU), which is described in Section 4.4. A description of the inputs, the output, and an associated status bit of the CALU follows.

a.Inputs. The CALU has two inputs One input is always provided by the 32-bit accumulator. The other input is provided by one of the following: The product-scaling shifter The input data-scaling shifter **Output**. Once the CALU performs an operation, it transfers the result to the 32-bit accumulator, which is capable of performing bit shifts of its contents. The output of the accumulator is connected to the 32-bit output data-scaling shifter. Through the output shifter, the accumulator's upper and lower 16-bit words can be individually shifted and stored to data memory.

c.Sign-extension mode bit. For many but not all instructions, the sign-extension mode bit (SXM), bit 10 of status register ST1, determines whether the CALU uses sign extension during its calculations. If SXM = 0, sign extension is suppressed. If SXM = 1, sign extension is enabled.

d.Accumulator

Once the CALU performs an operation, it transfers the result to the 32-bit accumulator, which can then perform single-bit shifts or rotations on its contents. Each of the accumulator's upper and lower 16-bit words can be passed to the output data-scaling shifter, where it can be shifted and then stored in data memory. The following describes the status bits and branch instructions associated with the accumulator.

Status bits. Four status bits are associated with the accumulator:

Carry bit (C). C (bit 9 of status register ST1) is affected during: _ Additions to and subtractions from the accumulator: C = 0 When the result of a subtraction generates a borrow When the result of an addition does not generate a carry (Exception: When the ADD instruction is used with a shift of 16 and no carry is generated, the ADD instruction has no effect on C.) C = 1 When the result of an addition generates a carryWhen the result of a subtraction does not generate a borrow (Exception: When the SUB instruction is used with a shift of 16 and no borrow is generated, the SUB instruction has no effect on C.) Single-bit shifts and rotations of the accumulator value. During a left shift or rotation, the MSB of the accumulator is passed to C; during a right shift or rotation, the LSB is passed to C. Overflow mode bit (OVM). OVM (bit 11 of status register ST0) determines how the accumulator reflects arithmetic overflows. When the processor is in overflow mode (OVM = 1) and an overflow occurs, the accumulator is filled with one of two specific values: If the overflow is in the positive direction, the accumulator is filled with its most positive value (7FFF FFFFh).If the overflow is in the negative direction, the accumulator is filled with its most negative value (8000 0000h). Overflow flag bit (OV). OV is bit 12 of status register ST0. When no accumulator overflow is detected, OV is latched at 0. When overflow (positive or negative) occurs, OV is set to 1 and latched.Test/control flag bit (TC). TC (bit 11 of status register ST1) is set to 0 or 1 depending on the value of a tested bit. In the case of the NORM instruction, if the exclusive-OR of the two MSBs of the accumulator is true, TC is set to 1.A number of branch instructions are implemented, based on the status of bits C, OV, and TC, and on the value in the accumulator (as compared to 0).

e.Output Data-Scaling Shifter

The output data-scaling shifter (output shifter) has a 32-bit input connected to the 32-bit output of the accumulator and a 16-bit output connected to the data bus. The shifter copies all 32 bits of the accumulator and then performs a left shift on its content; it can be shifted from zero to seven bits, as specified in the corresponding store instruction. The upper word (SACH instruction) or lower

word (SACL instruction) of the shifter is then stored to data memory. The content of the accumulator remains unchanged. When the output shifter performs the shift, the MSBs are lost and the LSBs are

zero filled. Figure 4–7 shows an example in which the accumulator value is shifted left by four bits and the shifted high word is stored to data memory.Figure 4–8 shows the same accumulator value shifted left by six bits and the shifted low word stored.

*5.4.3. **Auxiliary Register Arithmetic Unit (ARAU)***

The CPU also contains the ARAU, an arithmetic unit independent of the CALU. The main function of the ARAU is to perform arithmetic operations on eight auxiliary registers (AR7 through AR0) in parallel with operations occurring in the CALU. The eight auxiliary registers (AR7–AR0) provide flexible and powerful indirect addressing. Any location in the 64K data memory space can be accessed using a 16-bit address contained in an auxiliary register. To select a specific auxiliary register, load the 3-bit auxiliary register pointer (ARP) of status register ST0 with a value from 0 through 7. The ARP can be loaded as a primary operation by the MAR instruction (which only performs modifications to the auxiliary registers and the ARP), or by the LST instruction (which can load a data-memory value to ST0 by way of the data read bus,

DRDB). The ARP can be loaded as a secondary operation by any instruction that supports indirect addressing. The register pointed to by the ARP is referred to as the current auxiliary register or current AR. During the processing of an instruction, the content of the current auxiliary register is used as the address where the data-memory access will take place. The ARAU passes this address to the data-read address bus (DRAB) if the instruction requires a read from data memory; or, it passes the address to the data-write address bus (DWAB) if the instruction requires a write to data memory. After the instruction uses the data value, the contents of the current auxiliary register can be incremented or decremented by the ARAU, which implements unsigned 16-bit arithmetic.

1.ARAU Functions

The ARAU performs the following operations: Increments or decrements an auxiliary register value by 1 or by an index amount (by way of any instruction that supports indirect addressing) Adds a constant value to an auxiliary register value (ADRK instruction) or subtracts a constant value from an auxiliary register value (SBRK instruction). The constant is an 8-bit value taken from the eight LSBs of the instruction word. Compares the content of AR0 with the content of the current AR and puts the result in the test/control flag bit (TC) of status register ST1 (CMPR instruction). The result is passed to TC by way of the data write bus (DWEB). Normally, the ARAU performs its arithmetic operations in the decode phase of the pipeline (when the instruction specifying the operations is being decoded). This allows the address to be generated before the decode phase of the next instruction. There is an exception to this rule: During processing of the NORM instruction, the auxiliary register and/or ARP modification is done during the execute phase of the pipeline.

2.Auxiliary Register Functions

In addition to using the auxiliary registers to reference data-memory addresses, you can use them for other purposes. For example, you can: Use the auxiliary registers to support conditional branches, calls, and returns by using the CMPR instruction. This instruction compares the content of AR0 with the content of the current AR and puts the result in the test/control flag bit (TC) of status register ST1.

 Use the auxiliary registers for temporary storage by using the LAR instruction to load values into the registers and the SAR instruction to store AR values to data memory Use the auxiliary registers as software counters, incrementing or decrementing them as necessary

5.6. Addressing modes

The various addressing modes are

1.Direct addressing mode

2.Indirect addressing mode

3.Immediate Addressing Mode

5.6.1. Immediate Addressing Mode

In the immediate addressing mode, the instruction word contains a constant to be manipulated by the instruction. The two types of immediate addressing modes are:

1.**Short-immediate addressing.** Instructions that use short-immediate addressing have an 8-bit, 9-bit, or 13-bit constant as an operand. Short-immediate instructions require a single instruction word, with the constant embedded in that word.

Example:RPT #99 ;Execute the instruction that follows RPT 100 times

1 0 1 1 1 0 1 1	0 1 1 0 0 0 1 1
RPT opcode for immediate addressing	8-bit constant = 99

2.Long-immediate addressing. Instructions that use long-immediate addressing have a 16-bit constant as an operand and require two instruction words. The constant is sent as the second instruction word. This 16-bit value can be used as an absolute constant or as a 2s-complement value. In Example , the immediate operand is contained as a part of the RPT instruction word. For this RPT instruction, the instruction register will be loaded with the value Immediate operands are preceded by the symbol #.

Example:ADD #16384,2 ;Shift the value 16384 left by two bits ;and add the result to the accumulator

5.6.2. Direct Addressing Mode

In the direct addressing mode, data memory is addressed in blocks of 128 words called data pages. The entire 64K of data memory consists of 512 data pages labeled 0 through 511, as shown in Figure 6–3. The current data page is determined by the value in the 9-bit data page pointer (DP) in status register ST0. For example, if the DP value is 0 0000 00002, the current data page is If the DP value is 0 0000 00102, the current data page is 2. In addition to the data page, the processor must know the particular word being referenced on that page. This is determined by a 7-bit offset .The offset is supplied by the seven least significant bits (LSBs) of the IR register . Instruction Register (IR) Contents in Direct Addressing Mode instruction register, which holds the opcode for the next instruction to be executed. In direct addressing mode, the contents of the instruction register has the format

Instruction Register (IR) Contents in Direct Addressing Mode

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

8 MSBs 0 7 LSBs

8 MSBs Bits 15 through 8 indicate the instruction type (for example, ADD) and also contain any information regarding a shift of the data value to be accessed by the instruction.

0 Direct/indirect indicator. Bit 7 contains a 0 to define the addressing mode as direct.

7 LSBs Bits 6 through 0 indicate the offset for the data-memory address referenced by the instruction.

To form a complete 16-bit address, the processor concatenates the DP value and the seven LSBs of the instruction register.The DP supplies the nine most significant bits (MSBs) of the address (the page number), and the seven LSBs of the instruction register supply the seven LSBs of the address (the offset). For example, to access data address 003Fh,specify data page 0 (DP = 0000 0000 0) and an offset of 011 1111. Concatenating the DP and the offset produces the 16-bit address 0000 0000 0011 1111, which is 003Fh or decimal 63.

Generation of Data Addresses in Direct Addressing Mode

7 LSBs from IR

16-bit data-memory address

All 9 bits from DP

Data page pointer (DP)

Page (9 MSBs) Offset (7 LSBs)

Instruction register (IR)

9 bits 8 MSBs 0 7 LSBs

1.Using Direct Addressing Mode

When you use direct addressing mode, the processor uses the DP to find the data page and uses the seven LSBs of the instruction register to find a particular address on that page. Always do the following:

a. **Set the data page.** Load the appropriate value (from 0 to 511) into the DP. The DP register can be loaded by the LDP instruction or by any instruction that can load a value to ST0. The LDP instruction loads the DP directly without affecting the other bits of ST0, and it clearly indicates the value loaded into the DP. For example, to set the current data page to 32 (addresses 1000h–107Fh), you can use:

Example:LDP #32 ;Initialize data page pointer

b. **Specify the offset.** Supply the 7-bit offset as an operand of the instruction. For example, if you want the ADD instruction to use the value at the second address of the current data page, you would write:

ADD 1h ;Add to accumulator the value in the current ;data page, offset of 1.

Do not have to set the data page prior to every instruction that uses direct addressing. If all the instructions in a block of code access the same data page, you can simply load the DP at the front of the block. However, if various data pages are being accessed throughout the block of code, be sure the DP is changed whenever a new data page should be accessed.

Examples of Direct Addressing

Example:. Using Direct Addressing with ADD (Shift of 0 to 15)

LDP #4 ;Set data page to 4 (addresses 0200h–027Fh).

ADD 9h,5 ;The contents of data address 0209h are ;left shifted 5 bits and added to the

;contents of the accumulator.

7 LSBs from IR 16-bit data address 0209h

All 9 bits from DP DP = 4 Instruction register (IR)

0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 1

ADD opcode Shift of 5

0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 9h

In Example the ADD instruction references a data memory address that is generated as shown following the program code. For any instruction that performs a shift of 16, the shift value is not embedded directly in the instruction word; instead, all eight MSBs contain an opcode that not only indicates the instruction type, but also a shift of 16. The eight MSBs of the instruction word indicate an ADD with a shift of 16.

5.6.3. Indirect Addressing Mode

Eight auxiliary registers (AR0–AR7) provide flexible and powerful indirect addressing. Any location in the 64K data memory space can be accessed using a 16-bit address contained in an auxiliary register.

1. Current Auxiliary Register

To select a specific auxiliary register, load the 3-bit auxiliary register pointer (ARP) of status register ST0 with a value from 0 to 7. The ARP can be loaded as a primary operation by the MAR instruction or by the LST instruction. The ARP can be loaded as a secondary operation by any instruction that supports indirect addressing. The register pointed to by the ARP is referred to as the current auxiliary register or current AR. During the processing of an instruction, the content of the current auxiliary register is used as the address at which the data-memory access occurs. The ARAU passes this address to the data-read address bus (DRAB) if the instruction requires a read from data memory, or it passes the address to the data-write address bus (DWAB) if the instruction requires a write to data memory. After the instruction uses the data value, the contents of the current auxiliary register can be incremented or decremented by the ARAU, which implements unsigned 16-bit arithmetic. Normally, the ARAU performs its arithmetic operations in the decode phase of the pipeline (when the instruction specifying the operations is being decoded). This allows the address to be generated before the decode phase of the next instruction. There is an exception to this rule: during processing of the NORM instruction, the auxiliary register and/or ARP modification is done during the execute phase of the pipeline.

2. Indirect Addressing Options

The ’C24x provides four types of indirect addressing options:

a.No increment or decrement. The instruction uses the content of the current auxiliary register as the data memory address but neither increments nor decrements the content of the current auxiliary register.

b.**Increment or decrement by 1.** The instruction uses the content of the current auxiliary register as the data memory address and then increments or decrements the content of the current auxiliary register by one.

c.**Increment or decrement by an index amount.** The value in AR0 is the index amount. The instruction uses the content of the current auxiliary register as the data memory address and then increments or decrements the content of the current auxiliary register by the index amount. Indirect Addressing Mode

d.Increment or decrement by an index amount using reverse carry. The value in AR0 is the index amount. After the instruction uses the content of the current auxiliary register as the data-memory address, that content is incremented or decremented by the index amount. The addition and subtraction process is accomplished with the carry propagation reversed for fast Fourier transforms (FFTs).

e.Operand Option Example

* No increment or decrement **LT *** loads the temporary register (TREG) with the content of the data memory address referenced by the current AR.

*+ Increment by 1 **LT *+** loads the temporary register (TREG) with the content of the data memory address referenced by the current AR and then adds 1 to the content of the current AR.

*– Decrement by 1 **LT *–** loads the temporary register (TREG) with the content of the data memory address referenced by the current AR and then subtracts 1 from the content of the current AR.

*0+ Increment by index amount **LT *0+** loads the temporary register (TREG) with the content of the data memory address referenced by the current AR and then adds the content of AR0 to the content of the current AR.

*0– Decrement by index amount **LT *0–** loads the temporary register (TREG) with the content of the data memory address referenced by the current AR and then subtracts the content of AR0 from the content of the current AR.

*BR0+ Increment by index amount, adding with reverse carry

LT *BR0+ loads the temporary register (TREG) with the content of the data memory address referenced by the current AR and then adds the content of AR0 to the content of the current AR, adding with reverse carry propagation.

*BR0– Decrement by index amount, subtracting with reverse carry

LT *BR0– loads the temporary register (TREG) with the content of the data memory address referenced by the current AR and then subtracts the content of AR0 from the content of the current AR, subtracting with bit reverse carry propagation.

All increments or decrements are performed by the auxiliary register arithmetic unit (ARAU) in the same cycle during which the instruction is being decoded in the pipeline.

The bit-reversed indexed addressing allows efficient I/O operations by resequencing the data points in a radix-2 FFT program. The direction of carry propagation in the ARAU is reversed when the address is selected, and AR0 is added to or subtracted from the current auxiliary register. A typical use of this addressing mode requires that AR0 be set initially to a value corresponding to half of the array's size, and further, that the current AR value be set to the base address of the data (the first data point).

f. Next Auxiliary Register

In addition to updating the current auxiliary register, a number of instructions can also specify the next auxiliary register or next AR. This register will be the current auxiliary register when the instruction execution is complete. The instructions that allow you to specify the next auxiliary register load the ARP with a new value. When the ARP is loaded with that value, the previous ARP value is loaded into the auxiliary register pointer buffer (ARB).

Example:MAR*,AR1 ;Load the ARP with 1 to make AR1 the ;current auxiliary register.

LT *+,AR2 ;AR2 is the next auxiliary register ;Load the TREG with the content of the

;address referenced by AR1, add one to ;the content of AR1, then make AR2 the

;current auxiliary register.

MPY* ;Multiply TREG by content of address ;referenced by AR2.

5.7. Event Manager

Event Manager (EV) Functional Blocks

All devices of the ’240x family, with the exception of the ’2402, have two event each other in terms of functionality and register mapping/bit definition. For the sake of brevity, only the functionality of EVA is explained. Minor differences (such as naming conventions and register addresses) are highlighted as appropriate.

Each EV module in the ’240x device contains the following functional blocks:

1.Two general-purpose (GP) timers.

2.Three compare units.

3. Pulse-width modulation (PWM) circuits that include space vector PWM circuits, dead-band generation units, and output logic .

4.Three capture units (described in section 6.8 on page 6-66).

5. Quadrature encoder pulse (QEP) circuit

6.Interrupt logic .

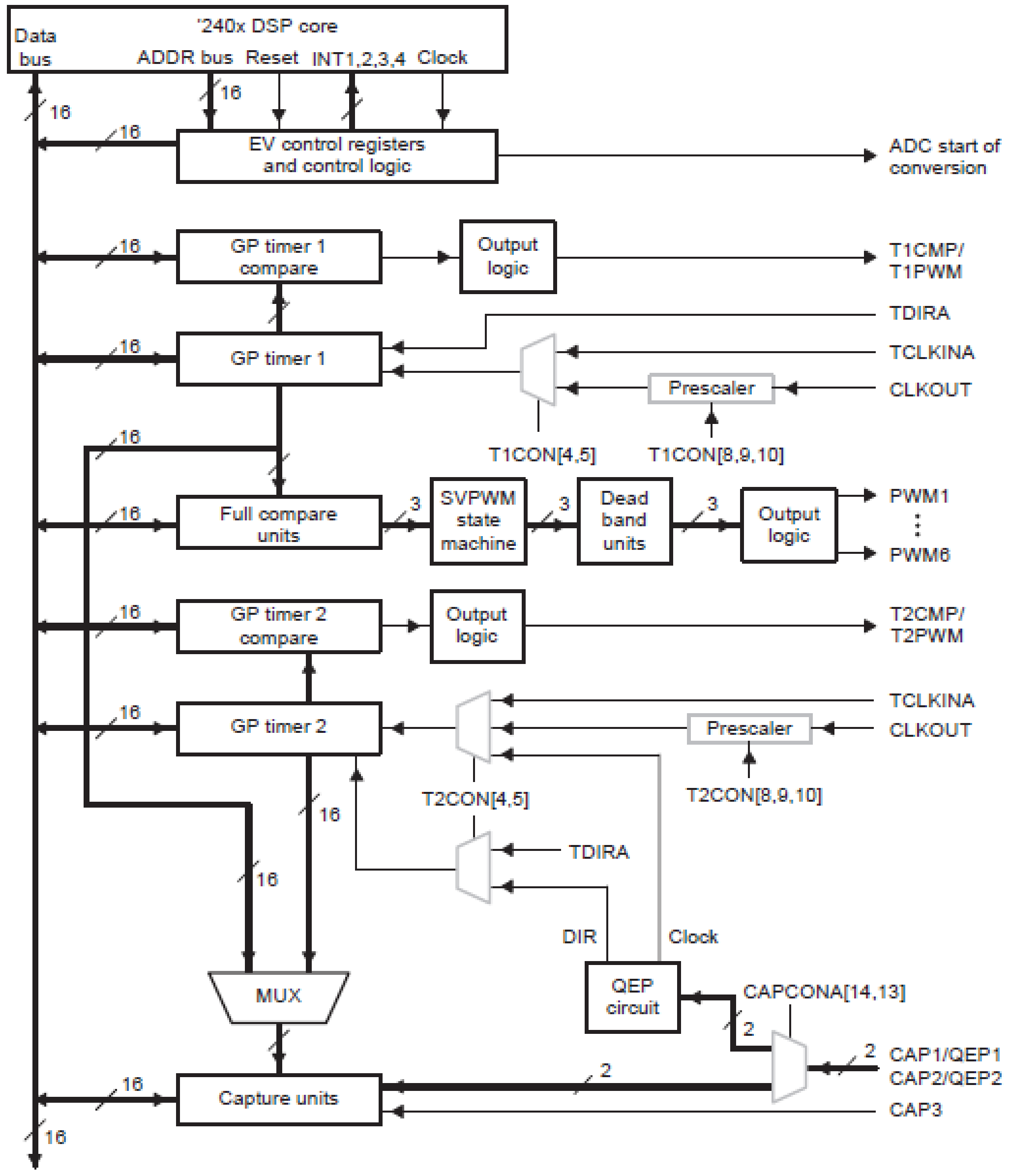


Fig.5.9. Block diagram Event manager

5.8.Simple Programs For PWM Generation

5.8.1. Algorithm

1. Include the 2407 register header file
2. Enable the PWM output pin using MCRA.
3. Load the data page of the even manager.
4. Enable the polarity of compare output and compare outputs using GPT con.
5. Initialize the timer 1 counter.

6. Load the timer 1 compare Register corresponding to the duty cycle.
7. Load the timer 1 period register corresponding to the switching frequency
8. Select the counting mode using timer 1 control register.
9. End.

5.8.2. Program

```
.include 2407 regs.h

        .text

        LDP    #0E1h

        SPLK  #1000h,MCRA

        LDP    #0E8h

        SPLK  #6042h.GPTCONA

        SPLK  #0000h, T1CNT

        SPLK  #800h, T1CMPR

        SPLK  #4000h, T1PR

        SPLK  #9042h, T1CON

H:      B    H
```

5.8.3. Speed control of PMDC motor control

Algorithm

Include the 2407 register header file.

Initialize even manager registers for timer underflow interrupt.

Configure required PWMs.

Load timer control registers.

Initialize minimum duty cycle and load value into compare register.

Check for underflow interrupt.

Read up button and down button switches.

Increment/decrement the PWM width correspondingly and check for limit.

Load the value to the compare register.

Repeat steps 6 to 9 for continuous variation of PWM.

Give the PWM pulse to the switches.

End.

Part A

1. What are the addressing modes of TMS320C24X processor.
2. What are the types of instructions of TMS320C24X processor.
3. Give 4 salient features of TMS320C24X processor.
4. What is event manager.
5. Give 4 applications of TMS320C24X processor.

Part B

6. Explain the architecture of TMS320C24X processor.

7. Write a simple program to control the speed of PMDC motor using TMS320C24X processor.

8. With a neat block diagram explain about event manager.