ELECTRONICS WORLD + WIRELESS WORLD

Interfacing with

Howard Hutchings

6 Digital filters

The rapid growth of micro-electronics and personal computers has meant that the characteristics of low-pass, band-pass and high-pass filters can be readily implemented in software form. This chapter describes the behaviour of a variety of digital filters in terms of convolution, impulse response, amplitude and phase response, using only elementary mathematics.

Much of the necessary groundwork – sampled-data signals and convolution - has already been outlined in Chapters 3 and 4. Examining the principles which support the design before presenting applications will encourage you to experiment.

Digital filters have numerous attractions: size, cost and flexibility favour the digital filter. Its response is independent of component tolerances, does not require alignment and is immune to temperature variations. Simply modifying the program changes the filter characteristics and it is possible to achieve results that are impractical using the analogue counterpart.

Certain filters are only possible using digital techniques. Alternatively, the characteristics of well-proven analogue designs – Butterworth, Chebychev – can be represented in digital form by means of the bilinear transform. In the interest of clarity, I shall concentrate initially on z-domain designs, believing that the added abstraction of the bilinear transform increases confusion, in addition to generating tedious algebra.

Figure 6.1 shows how an A-to-D converter, computer and D-to-A converter may be connected to achieve the necessary processing. The input peripheral is preceded by an anti-aliasing filter, while the output peripheral drives a simple low-pass filter which reconstructs the sampled signal into analogue form.

Peripheral hardware requirements

Fast analogue-to-digital conversion is straightforward using the Blue Chip Technology general-purpose I/O card ACM-44, shown in Figure 6.2. As usual, the base address is selectable in the prototyping region, thereby avoiding bus contention. The analogue input section features a software-controlled, 16-channel single-ended or 8-channel differential multiplexer. Full-scale input voltage is link-selectable in the ranges 0–2.5 V, 0–5 V or 0–10 V. A-to-D conversion is achieved using the fast Analog Devices AD7820, an 8-bit half-flash converter.

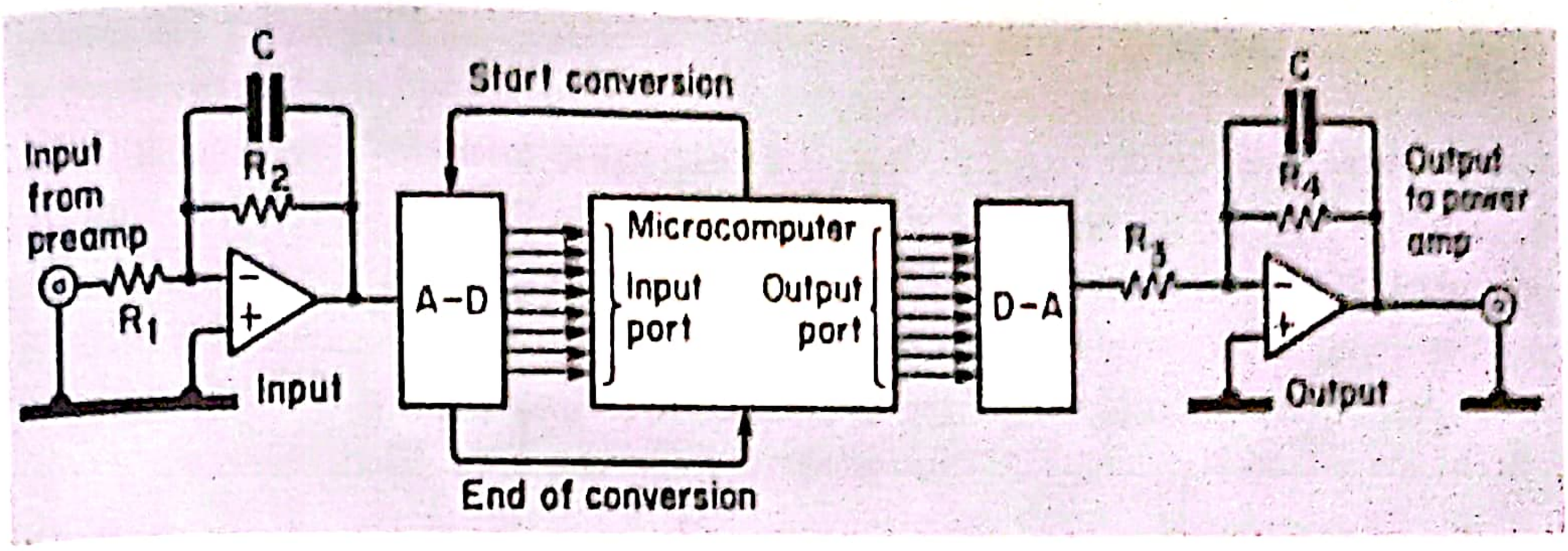


Figure 6.1 Representing an analogue filter by a digital technique. Low-pass filter at input avoids errors due to aliasing, while that at output smooths steps in processed signal

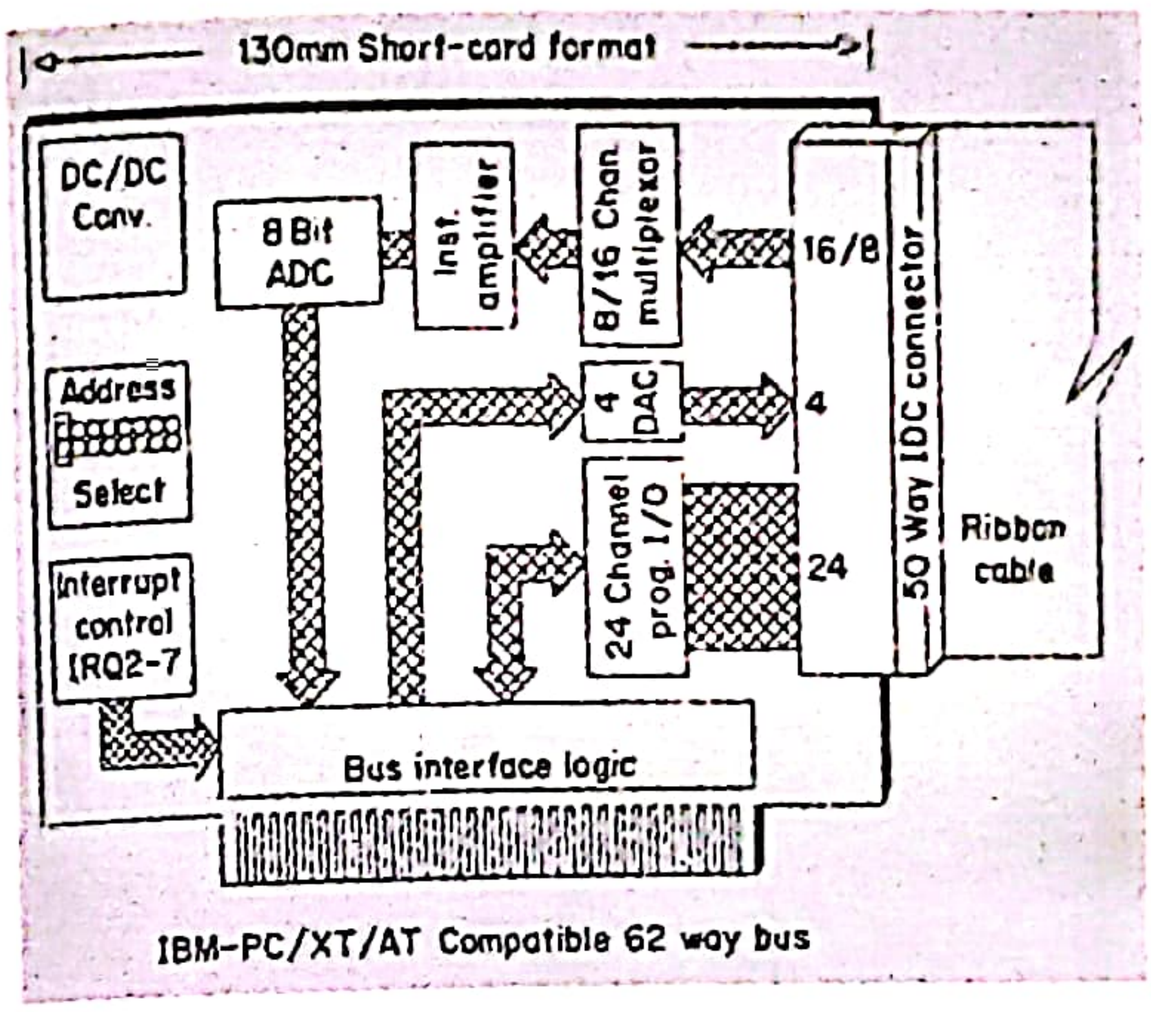


Figure 6.2 Blue Chip Technology general-purpose I/O card, used here for fast A-to-D conversion. The card is compatible with PC XT/AT computers

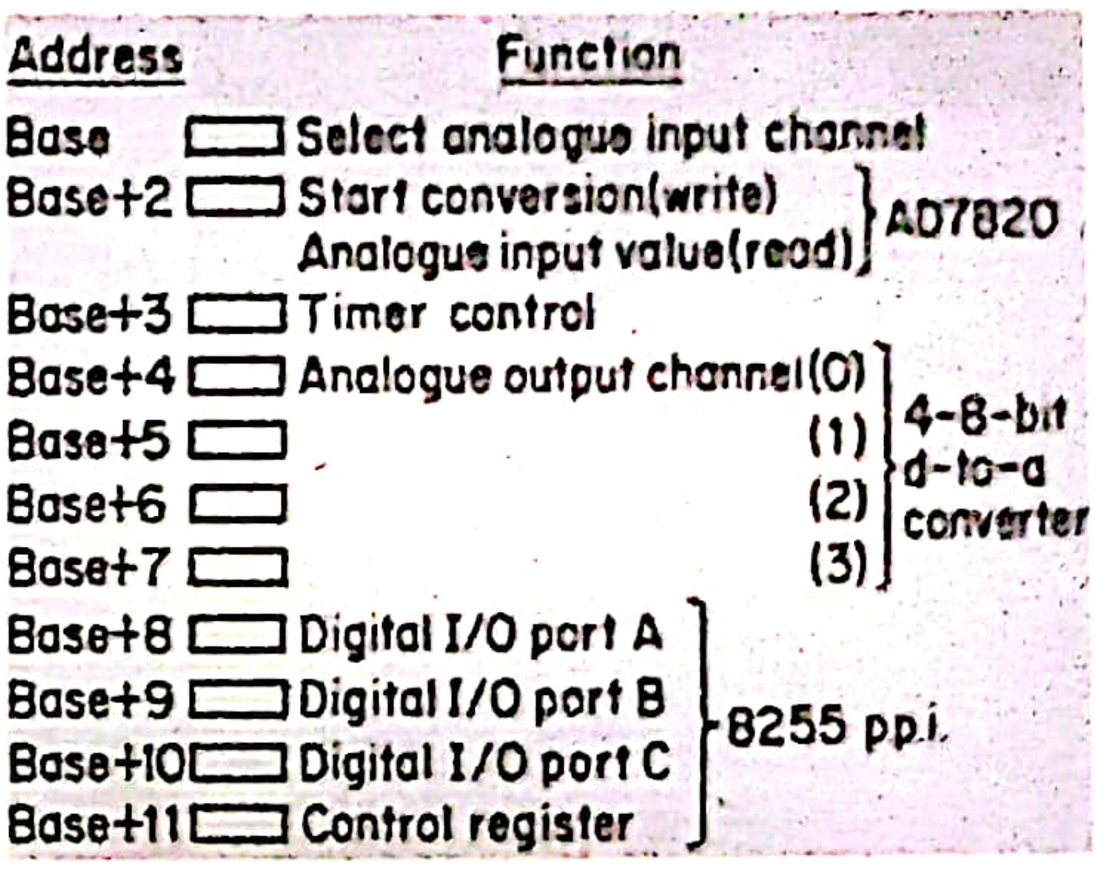


Figure 6.3 Port map and register model of I/O card

Software control is particularly elegant; strobing the A-to-D by writing to the input data port initiates conversion, which is completed in 1.56 μs, making testing for end-of-conversion unnecessary. Generous provision for D-to-A conversion is provided by four 8-bit converters, the output voltage being selectable in the ranges 5 V or 10 V full scale. In addition, the board provides a single 8255 PPI. The port map and register model is shown in Figure 6.3.

Auxiliary maths co-processor

Real-time digital filtering requires that a signal should be sampled, processed in some way and finally output to the real world before the next sample becomes available. Using a PC keeps the external hardware to a minimum, but relies very heavily on software for signal manipulation, which is traditionally slow.

Standard XT/PCs using 8086/8088 processors can add two numbers together

in about 1 μs and multiply in about 25 μs. The faster AT/PC, using the 80286 processor, performs the same addition roughly twice as quickly, while multiplication is more than seven times as fast. Despite these impressive computational figures it is possible to improve performance significantly by complementing the standard processor with an auxiliary maths co-processor. It is difficult to be precise about what speed improvement to expect – the actual calculation is likely to suggest 50 to 100 times as fast, but the program housekeeping and co-processor overheads are likely to reduce this advantage to the 5 to 20 times range.

The XT-compatible 8087 co-processor, or AT-compatible 80287 numeric data processor, is a 40-pin chip which simply plugs into the socket provided on the main system board. As its name suggests, the co-processor combines with the architecture of the host processor to augment both the speed and precision of numerical computations. The increased speed is due to the more extensive register model, from which the software builds up the calculation. Increased accuracy is due to the operands being held in 80 bit-wide registers, in a format called temporary real, precise enough to guarantee 18-decimal digit accuracy. Operational details of these registers are beyond the scope of this book – to the C programmer the co-processor is transparent, no special functions being required; simply write your program and leave the host processor to worry about the details.

Conversion of sinusoidal signals

This program, Listing 6.1, is designed to be used with the digital signal-processing system shown in Figure 6.1. Sinusoidal signals in the range 0–5 V are input through the A-to-D converter and processed in real time by the program in the computer, before being output in real-time through the D-to-A converter, conditioned in the range 0–5 V full scale. The processed signal is finally fed into a power amplifier and speaker system.

As a confidence check, or benchmark, it is worthwhile processing data from the input peripheral through the computer to the output peripheral, with no signal conditioning at all.

Although this program may not appear to do a great deal, it should be regarded as a useful prototype, checking the hardware connections together with the software. As a matter of routine, I always try to keep it simple, before checking out more ambitious systems and software. All too often I find that time spent confirming the obvious pays dividends later on, when things don't work as expected.

Successful digital signal processing requires the sampling frequency to be at least twice the highest signal frequency present. This is an important parameter, the practical limits being determined by the choice of software and the characteristics of the A-to-D converter. May I remind you that C is fast, and running Listing 6.1 with the aid of a maths co-processor makes it very fast, resulting in a sampling frequency of 40 kHz. The effect of digitally processing a

Listing 6.1 Sinusoidal digital signal processing

```
/**************************
* SIMPLE DIGITAL SIGNAL *
* PROCESSOR 25 MICRO-SEC *
* SAMPLING INTERVAL *
**************************/
#include<stdio.h>
#include<conio.h>
#define BASE 768
main()
{
unsigned int contents;
outp(BASE,1);
/*-------------
SELECT I/P CHANNEL
-------------*/
start:outp(BASE+2,0)
/*----------
START CONVERSION
----------*/
contents = inp(BASE+2);
/*---------
READ I/P PORT
---------*/
outp(BASE+4,contents);
/*----------
WRITE TO O/P PORT
----------*/
goto start;
}
```

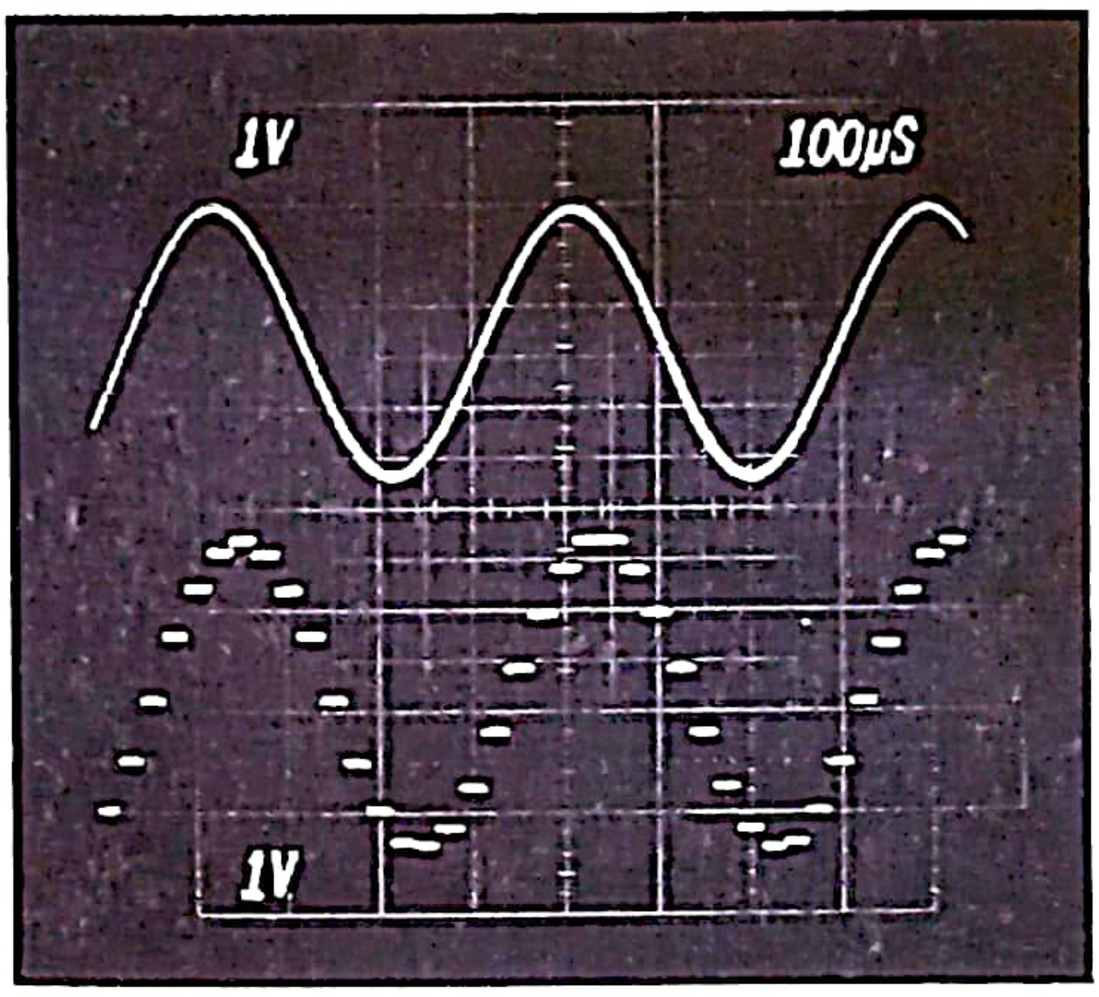


Figure 6.4 Output of the D-to-A converter with a 2.5 kHz sine input and sampling frequency of 40 kHz

2.5 kHz sine wave using this system is shown in Figure 6.4, in which the 25 μs sampling interval is clearly visible.

Programming is often an iterative process. In this case, the strategy is first to make the program work, then to make it well-behaved and finally to make it fast. Listing 6.1 features the goto statement, although repetition could have been achieved with an infinite for loop as indicated in Chapter 1. Replacing the unconditional jump structure as suggested resulted in no measurable improvement in sampling rate.

It is worth spending a little time investigating the characteristics of the digital approximation to the input sine wave. Figure 6.5 shows the output from the D-to-A in greater detail. Each step of the 2.5 kHz sine wave is composed of a sample of duration 25 μs, equivalent to the sampling rate of the digital system. Using a spectrum analyser we may determine the harmonic content of the staircase output – the Fourier spectrum is shown in Figure 6.6. Typically, there is a substantial component at 2.5 kHz, which routinely would be unique. However, the stepped nature of the sine wave introduces significant components at higher

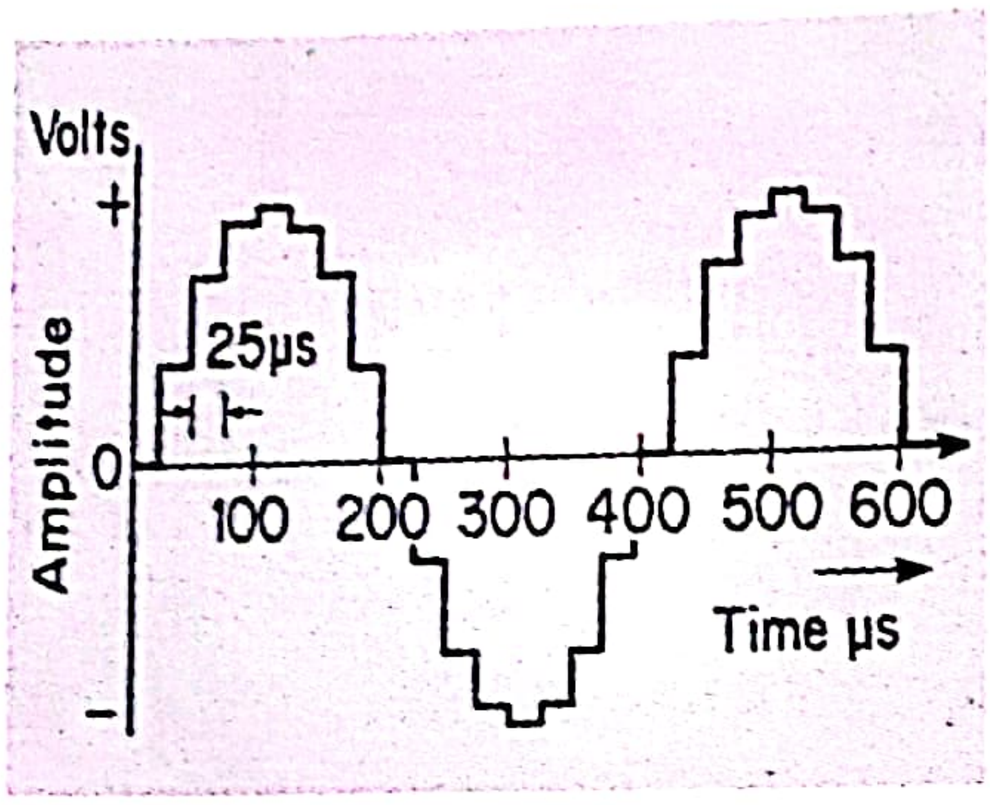


Figure 6.5 Output of the A-to-D in greater detail. Each sample is 25 μs in duration, equivalent to the period of the sampling frequency

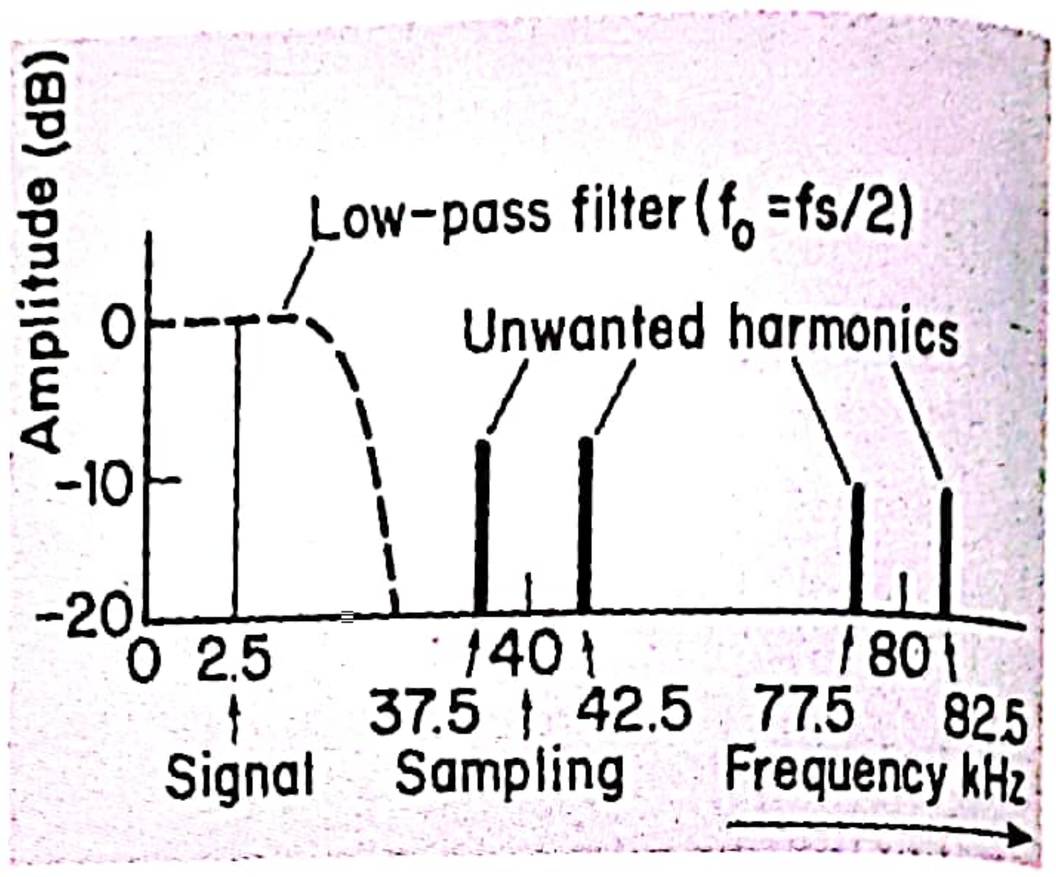


Figure 6.6 Spectrum of the A-to-D output. Stepped nature of output generates harmonics, which are removed by the low-pass filter.

frequencies. These appear as sidebands symmetrically displaced around integer multiples of the sampling frequency. Examine Figure 6.6 carefully and you will see how the low-pass filter connected across the output of the D-to-A removes these unwanted harmonics, leaving the required sinusoid.

Before we become immersed in the fine detail of digital filtering, a few words on strategy would be appropriate. Expressed simply, the object is to use digital methods to simulate the filtering methods previously achieved using combinations of resistance, inductance and capacitance. Progress is made by concentrating on the intended system response, conveniently described by the transfer function. In this way we avoid being distracted by the electrical characteristics of the individual components.

A digital filter is usually a subroutine of the main program, one which accepts sample values, processes them in real time and then returns samples representing the filtered output. The algorithms used in the subroutine determine the particular parameters of the filter, such as cut-off frequency, Q-factor and rate of roll-off.

Care must be exercised when interpreting the frequency characteristics of digital filters. The periodic nature of linear sampled-data systems ensures that the frequency-response curves shown in Figure 6.7 will repeat indefinitely at multiples of the sampling frequency. Consequently the terms low-pass, band-pass and high-pass should be characterized in terms of the cyclic or signal frequency divided by the sampling frequency. Conventionally, the horizontal axis is labelled in rads^{-1} but, since most electronics engineers think in hertz, I have annotated the horizontal axis as ωT radians, where $\omega T = 2\pi f_{\text{signal}}/f_{\text{sampling}}$.

Analogue and digital filters

As a more challenging problem we intend to use the personal computer and associated peripherals shown in Figure 6.1 as a first-order, low-pass digital filter. Our intention is to use digital signal-processing methods to simulate the transfer function of the familiar analogue filter shown in Figure 6.8. The deliberate

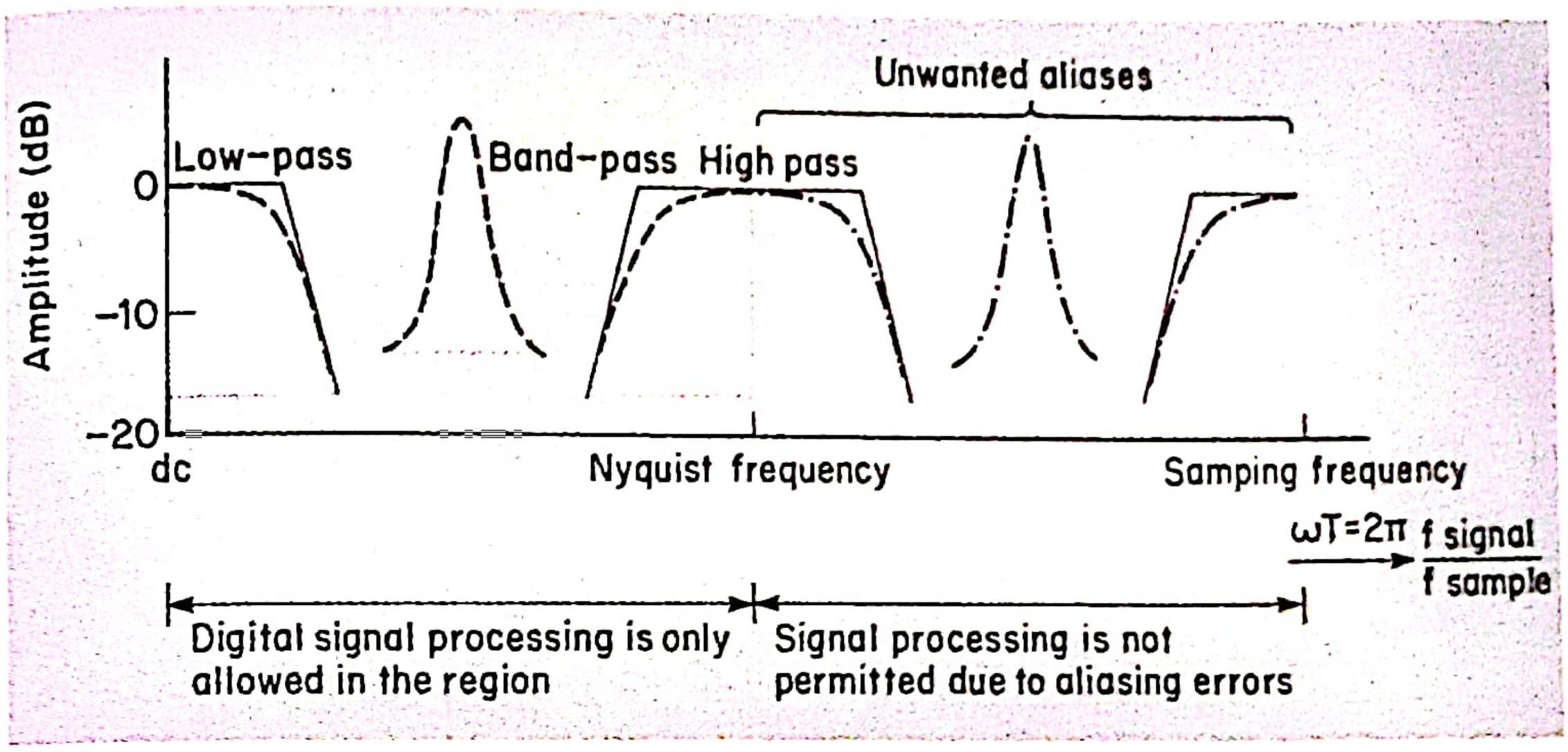


Figure 6.7 Response of three types of digital filter. Curves repeat at multiples of sampling frequency; processing not permitted beyond Nyquist frequency

adoption of a tried and tested analogue prototype is a productive way of making rapid progress using digital methods, because we can easily anticipate the likely form of the processed output.

The filter will be implemented in software by an algorithm within the computer which effectively convolves the sampled sequence $x(n)$ with the impulse response of the filter $h(n)$ to produce the processed output sequence $y(n)$. As illustrated previously, convolution is the name given to the ordered combination of multiplication and summation.

The analogue signal presented to the A-to-D will be processed into a series of discrete samples. To avoid aliasing it is necessary to sample at a rate which is at least twice the Nyquist frequency (twice the highest frequency present).

Suppose the sampled signal can be represented by the sequence of pulses $x(0)$, $x(1)$, $x(2)$, $x(3)$, $x(4)$, each of which is separated from its neighbour by the fixed interval T, the sampling time. Each sample pulse in turn will stimulate the filter, which responds with a train of weighted output pulses. If the memory of the filter is long enough, there will still be a vestige of the previous history of weighted outputs, even as the current pulse is being processed (Figure 6.9).

Our intention is to demonstrate the effects of processing a square wave when the fundamental frequency is close to the cut-off frequency of the filter. Referring to the characteristics of the analogue filter shown in Figure 6.8, it will be clear that beyond the break frequency the amplitude of the input signal is reduced at a rate of -20 dB/decade. The amplitude attenuation is accompanied by a progressive phase lag over the range $0.1f_1$ to $10f_1$, the latter approaching $-90°$ as the frequency tends towards infinity.

Many of the preliminaries have already been outlined in Chapters 4 and 5 of this book; we intend to build on these results and investigate the form of processed output obtained by convolving a sampled square wave with the impulse response shown in Figure 6.10.

To evaluate the rest of the terms in the sequence, continue moving the reversed

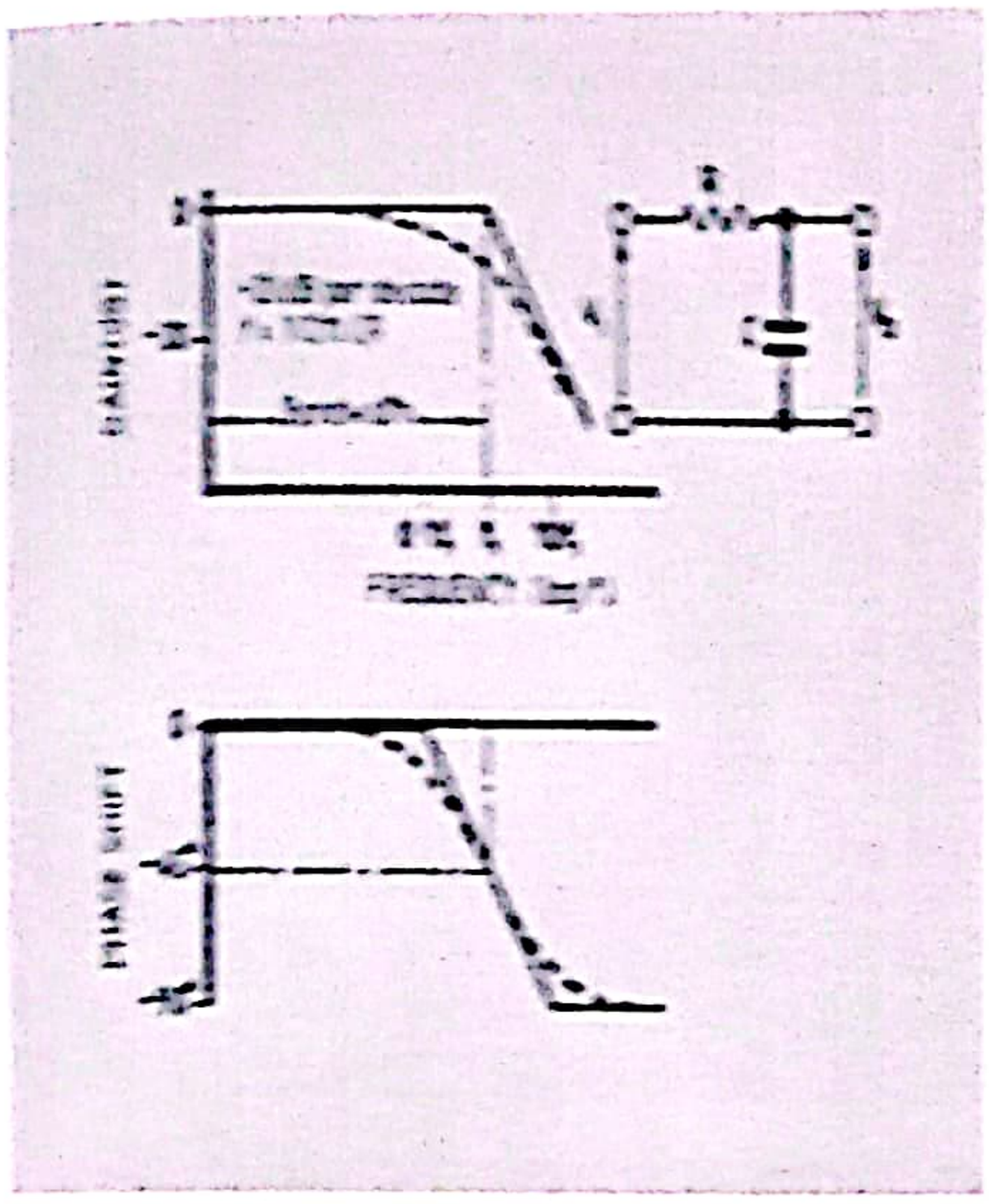


Figure 6.8 Simple, first-order, low-pass analogue filter has time constant CR seconds. Low frequencies are processed with negligible amplitude and phase modification

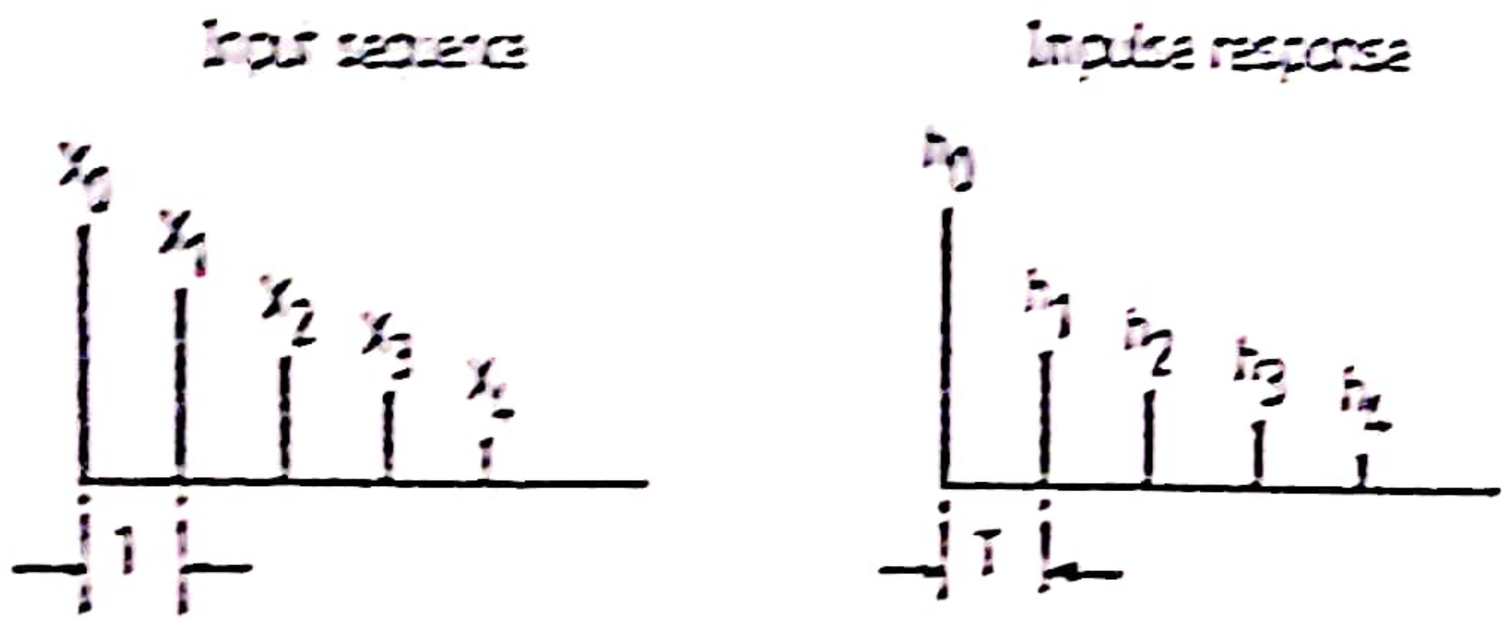


Figure 6.9 Sampled input is represented by sequence of pulses separated by fixed interval T. Linear processor characterized by impulse response sequence

impulse response to the right, until neither of the two sequences overlap. My results are

$y(0) = 0.5$
$y(1) = 0.75$
$y(2) = 0.875$
$y(3) = 0.9375$
$y(4) = 0.9688$
$y(5) = 0.9844$
$y(6) = 0.9922$
$y(7) = 0.9961$
$y(8) = 0.9980$
$y(9) = 0.9990$

These are plotted in Figure 6.11, in which the exponential nature of the output waveform suggests that the digital signal processing is similar to that produced by the simple low-pass analogue filter shown in Figure 6.8.

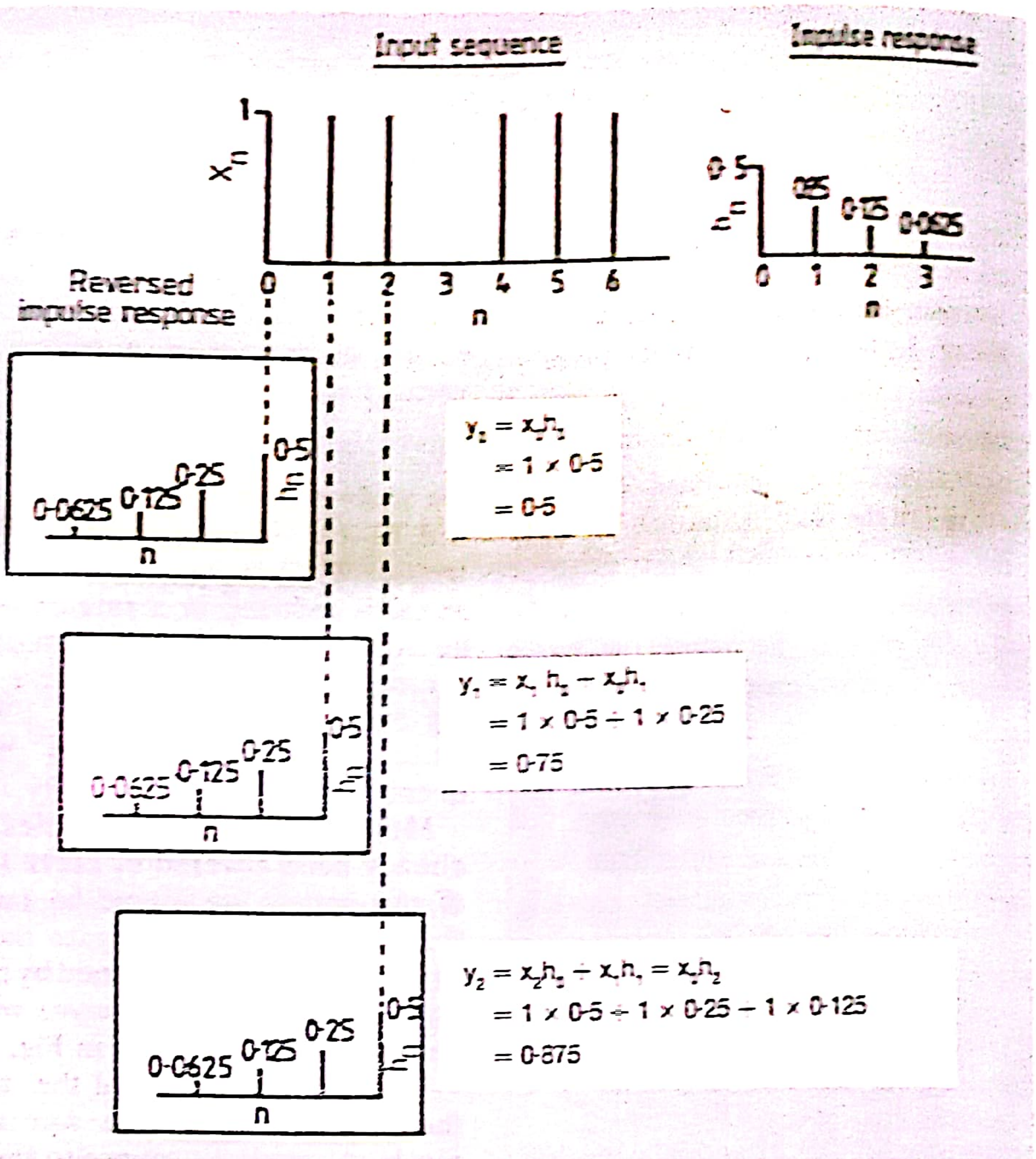


Figure 6.10 Graphical convolution of sampled square and impulse response is obtained by reversing impulse response under sample of current interest. Sum of coincident cross-products is convolution of that sample and impulse response of processor

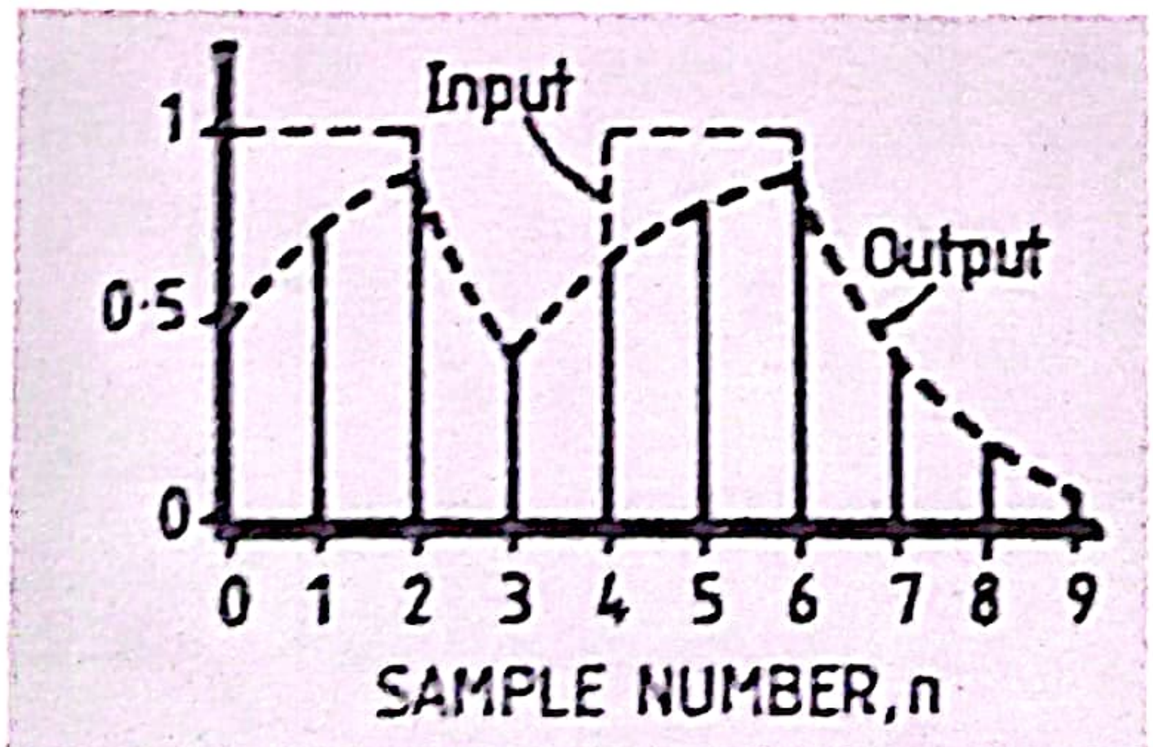


Figure 6.11 Signal processing reduces certain high-order harmonics and introduces some time delay. Remaining harmonics sum at output to produce familiar exponential response

Convolution in the time domain corresponds to multiplication in the frequency domain. This processing operation is equivalent to multiplying the frequency spectrum of the signal by the system frequency response, the product corresponding to a filtering operation. Much of this is vividly illustrated by the Fourier series description of the square wave, amplitude ± 1 V, together with the frequency response of the first-order, low-pass signal processor. Figure 6.12 shows the spectral characteristics, demonstrating the equivalence in processing operations in the time and frequency domains.

Figure 6.12(a) shows how the input square wave can be decomposed into a sinusoidal fundamental, together with an infinite number of odd (in phase) harmonics. Superimposing the spectral contents of the signal on those of the filter demonstrates that a significant number of harmonics are contained within the passband of the filter. Although the amplitude attenuation in this range is small, there is still appreciable phase delay; the overall effect is to produce a little exponential rounding of the leading and trailing edges of the processed output, as in Figure 6.12(b). Reducing the bandwidth of the filter while keeping the frequency of the input signal constant, as seen in Figure 6.12(c), results in significant amplitude attenuation and almost constant phase delay of 90°.

To implement the signal-processing characteristics of the analogue first-order, low-pass filter described in Figure 6.8, re-examine the form of the impulse response $h(n)$. This was simply the sequence 0.5, 0.25, 0.125, 0.0625, truncated after the fourth term for ease of calculation. We propose to use the mathematical

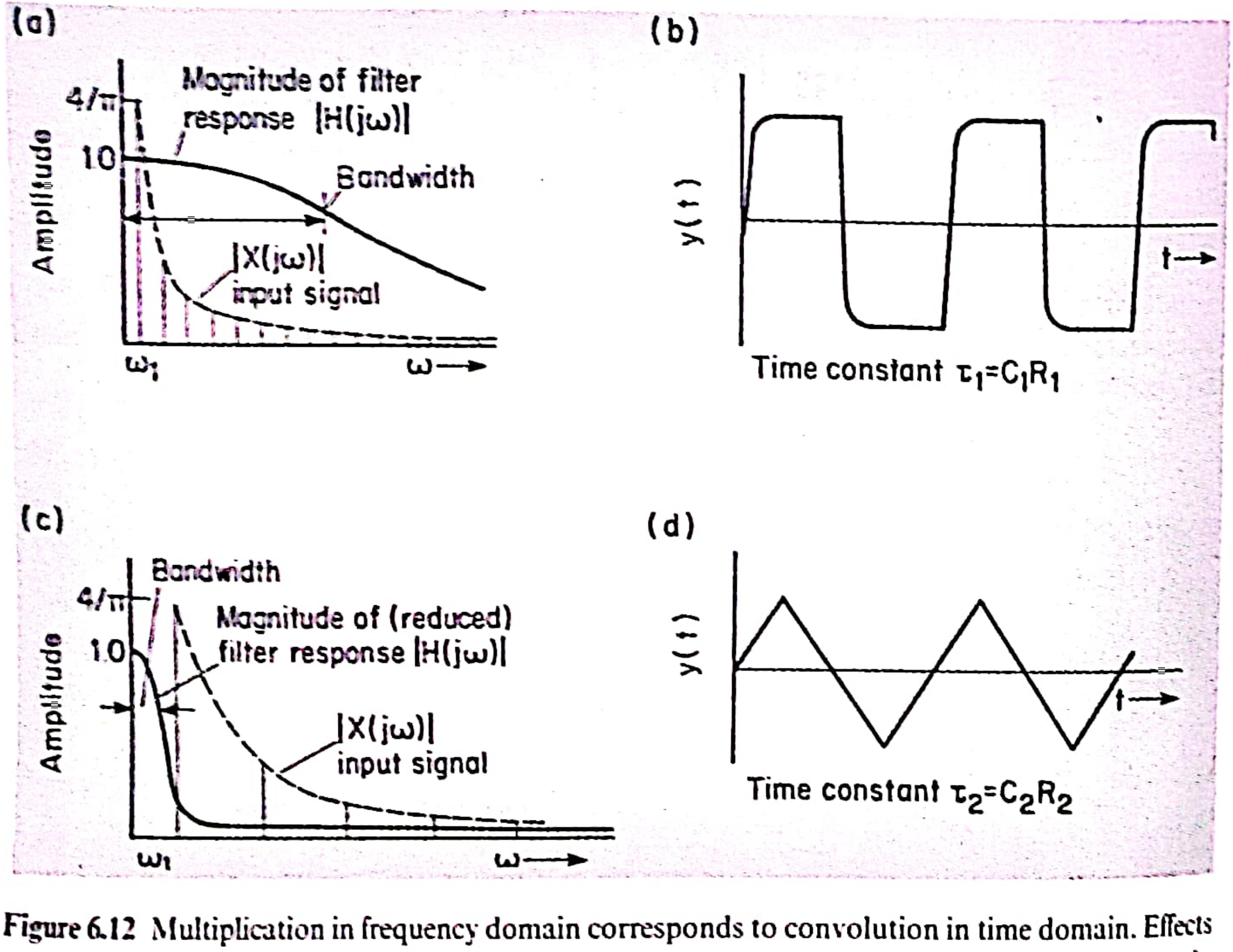


Figure 6.12 Multiplication in frequency domain corresponds to convolution in time domain. Effects of reducing bandwidth of filter while keeping input square-wave frequency constant are shown in frequency (a)(c) and time (b)(d) domains

models introduced in Chapter 3 to describe this behaviour more completely. The time-domain model of the impulse response of the filter is

$$h(t) = 0.5\delta(t) + 0.25\delta(t - T) + 0.125\delta(t - 2T) + 0.0625\delta(t - 3T) + \ldots$$

To establish the complex frequency domain model, we must take the Laplace transform of the impulse response:

$$H(s) = 0.5 + 0.25e^{-sT} + 0.125e^{-2sT} + 0.0625e^{-3sT} + \ldots$$

Finally, using the z-transform to model the delayed weighted samples, we can write

$$H(z) = 0.5z^{0} + 0.25z^{-1} + 0.125z^{-2} + 0.0625z^{-3} + \ldots$$

Recognizing that this is an infinite geometric series, we may express $H(z)$ in closed form:

$$H(z) = 0.5/(1 - 0.5z^{-1})$$

This is simply the transfer function $Y(z)/X(z)$ of the filter in the z-domain. The relationship between the processed output $Y(z)$ and the input sample $X(z)$ is given by

$$Y(z) = X(z)H(z)$$
$$Y(z) = 0.5X(z)/(1 - 0.5z^{-1})$$

Cross-multiplying gives the result:

$$Y(z)(1 - 0.5z^{-1}) = 0.5X(z)$$

So the recurrence relationship describing the behaviour of the filter is

$$y(n) = 0.5x(n) + 0.5y(n - 1)$$

where $x(n)$ is the current input, $y(n)$ the current output and $y(n - 1)$ the previous output. Figure 6.13 shows the processing operations.

Recursive software

A computer performs calculations such as these with ease. Realizing the filter recursively has the advantage that only a finite number of iterations are necessary to process the signal completely, despite the filter or system being characterized by an infinite impulse response.

Using Listing 6.2 in conjunction with the signal-processing system shown in Figure 6.1 produces a real-time low-pass digital filter, which samples every 160 μs. Figure 6.15 shows the effect of processing a square wave of frequency 250 Hz.

Comparing the processed output $y(n)$ obtained by recurrence with that obtained by using convolution indicates a certain lack of correlation after the fourth term, these relatively small numerical inconsistencies being due, in this case, to the truncated impulse response. However, this theoretical example serves

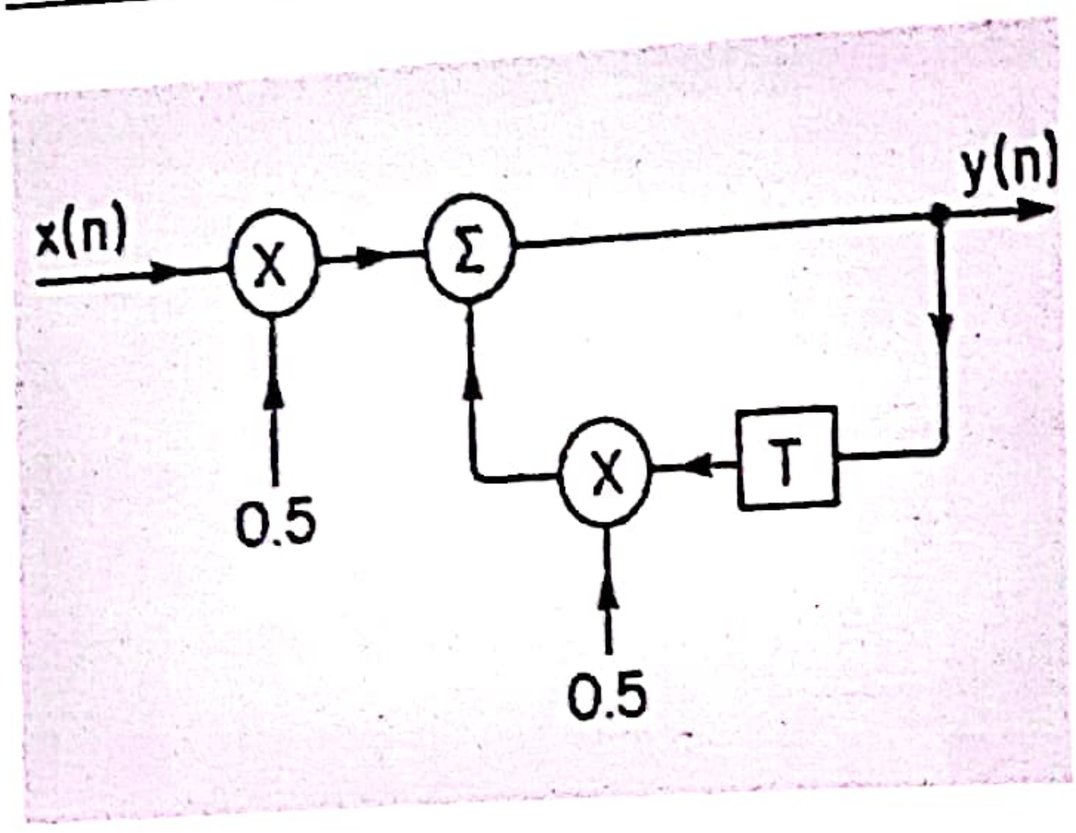


Figure 6.13 The delay element in the feedback loop retains the weighted processed output $y(n)$ for one sampling interval. It then releases it to the summation element, where it is added to the current weighted input. To produce the processed output $y(n) = 0.5x(n) + 0.5y(n-1)$

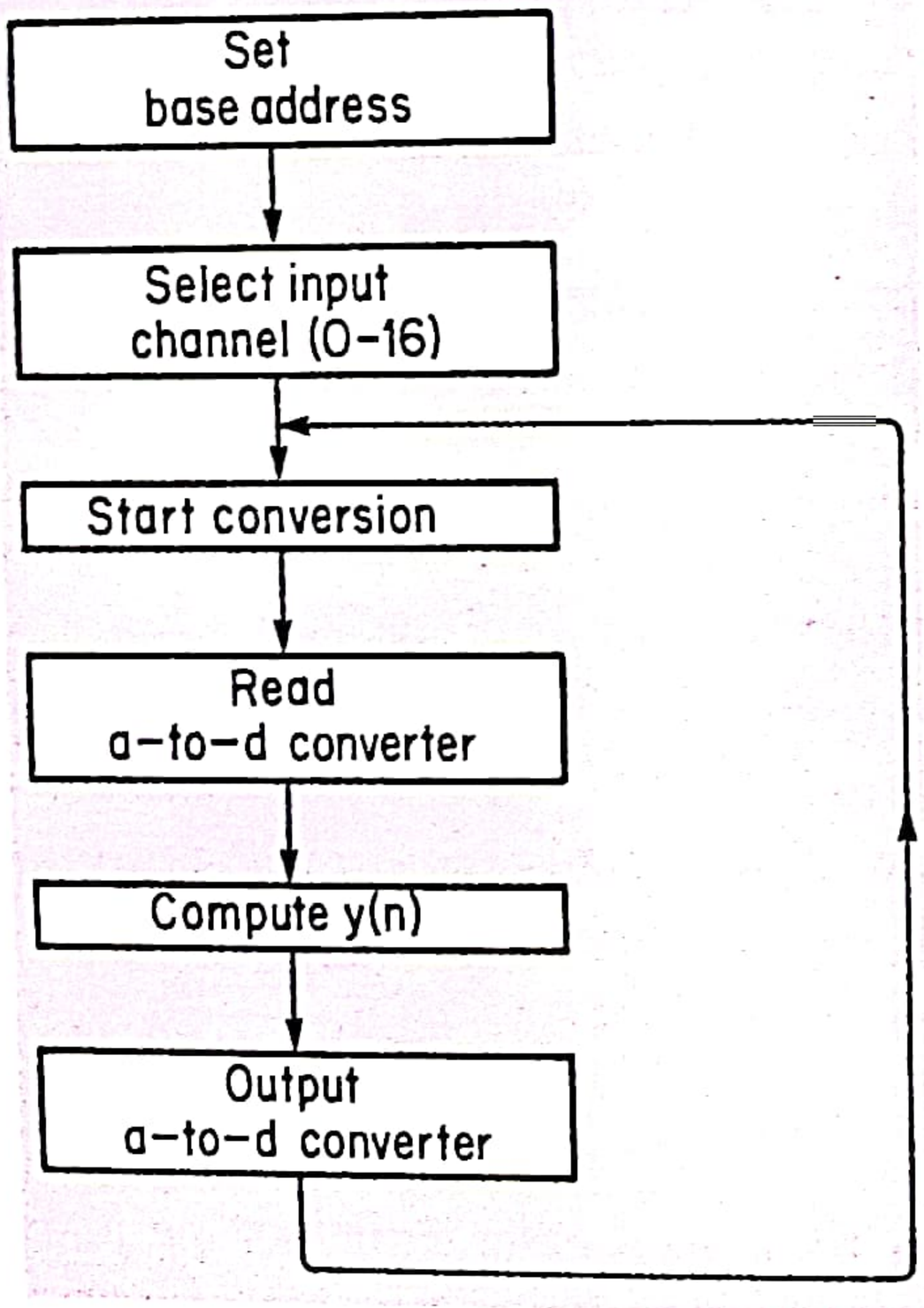


Figure 6.14 Flow chart for recursive realization of low-pass digital filter

Listing 6.2 Low-pass digital filter

```
/*********************************
* LOW-PASS DIGITAL FILTER *
* y(n) = 0.5x(n) + 0.5y(n - 1) *
*********************************/
#include<stdio.h>
#include<conio.h>
#define BASE 768
main()
{
unsigned int contents,output,old_output;
outp(BASE,1);
/*-------------
SELECT I/P CHANNEL
---------------*/
start:outp(BASE + 2,0);
/*-------------
START CONVERSION
-------------*/
contents = inp(BASE + 2);
/*----------
READ I/P PORT
----------*/
contents = 0.5 * contents;
output = contents + old_output;
outp(BASE + 4,output);
old_output = 0.5 * output;
/*--------------------------
RECURRENCE RELATIONSHIP
y(n) = 0.5 * x(n) + 0.5 * y(n - 1)
--------------------------*/
goto start;
}
```

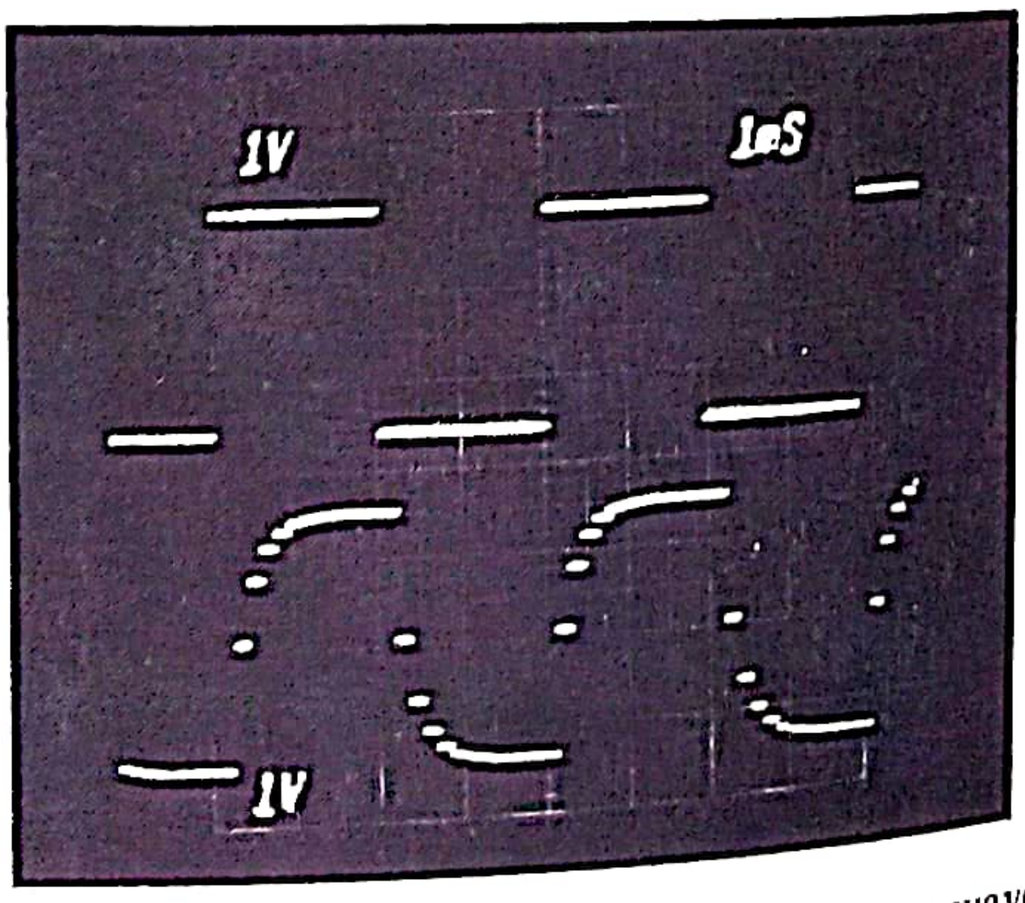


Figure 6.15 Oscilloscope display of square-wave input to A-to-D at 250 Hz and output from D-to-A. Sampling interval 160 μs

as a useful illustration, identifying a few of the practical hazards inherent in finite word arithmetic.

When the signal and multiplier coefficients are represented in terms of binary digits, any numerical processing is likely to be subject to error. A principal source

of inaccuracy is the integer output of the A-to-D converter, which may be in error by as much as ±0.2% of full-scale for an 8-bit converter. Clearly, 12-bit and 16-bit devices will improve any quantization error, possibly at the expense of conversion time.

Potentially more hazardous is the inaccurate specification of the coefficients of recursive filters, where rounding errors may result in the system poles moving outside the unit circle, resulting in instability and a badly behaved system. An acceptable approach is to specify multiplier coefficients to four or five decimal figures. Exceeding the dynamic range of the system will result in non-linear processing, a confused output and a collapse of the digital filter.

Poles and zeros

When the current output depends on past output samples in addition to current and past input samples, the filter is termed recursive. A digital filter may be realized in recursive form whenever the transfer function has poles at locations other than the origin of the z-plane. As we have seen, such a filter is characterized by an impulse response that contains an infinite number of terms. The recurrence relationship

$$y(n) = Ax(n) + By(n-1)$$

provides a useful recursive method of implementing a simple low-pass filter. To gain greater insight into the behaviour of the system it is necessary to augment our mathematical toolkit and discuss the concept of poles and zeros already hinted at in the development of transforms.

Z-domain model

The characteristics of a linear digital system may be described in terms of the transfer function $H(z)$. It is a general property of such systems that there are certain values of the complex variable z which make $H(z)$ tend to infinity, and values of z that make $H(z)$ zero. These values are known respectively as the poles and zeros.

Poles and zeros represent critical frequencies where something dramatic happens to the transfer function; as a result of poles and zeros, the function changes as z varies. Unfortunately, I cannot justify this statement without reference to a bookcase of relatively advanced mathematical text books, which is not my intention. However, if you are prepared to accept this in good faith, then what follows will provide a reasonable insight into the frequency response of digital filters.

No real system can have more zeros than poles. To ensure that the system remains stable, the designer must ensure that the poles are located on, or preferably inside, the unit circle. Clearly, the locations of the poles and zeros dictate the behaviour of the system. In an attempt to dispel this fog of

mathematical abstraction, consider what we already know. To model time advances or delays it is expedient to use the operator z. Applying the operator to the advertised recurrence relationship, we may write

$$Y(z) = AX(z) + Bz^{-1}Y(z)$$

So the transfer function $H(z)$ can be expressed as

$$H(z) = Y(z)/X(z) = Az/(z - B)$$

By inspection, the system has a single pole at $z = B$, together with a zero at the origin. Some perception of these rather evocative terms may be gained by plotting the magnitude of $H(z)$, so that it describes a surface. We do not propose to plot all the values, but rather to focus our attention on the value of z which makes $H(z)$ infinite – the pole. Visualize a rubber sheet stretched over the z-plane; when $z = B$ the surface goes shooting off to infinity (imagine a vaulting-pole placed on the plane at this point; the zero located at the origin serves to glue the surface down). An easy-to-understand geometrical illustration is given in Figure 6.16.

Frequency response and operator z

Previous use of the operator z to model time advances or delays placed no restriction on the shape of the time-shifted waveform. The complex variable $s = \alpha + j\omega$ in the expression $z = e^{sT}$ took care of this. In this particular application we are primarily interested in how a digital filter will process sinewaves; we may relax our original definition of s and discard the exponent α. This means that z may be viewed as a phasor defined by $z = e^{j\omega t}$. The transfer function $H(j\omega)$ represents a complex number and provides a mathematical model of the sinusoidal behaviour of the system.

Viewed as a sinusoidal signal processor the transfer function of the filter may be written as

$$H(j\omega) = Ae^{j\omega T}/(e^{j\omega T} - B)$$

or equivalently, using Euler's rule to express in real and imaginary parts:

$$H(j\omega) = A[\cos \omega T + j \sin \omega T]/\cos \omega T - B + j \sin \omega T$$

Describing $H(j\omega)$ as a complex number is simply a succinct method of describing the amplitude and phase characteristics in a single expression. To extract the magnitude of the frequency response we multiply $H(j\omega)$ by $H^*(j\omega)$, its complex conjugate, before taking the square root of the product. For this particular example, the magnitude of the frequency response is given by the expression

$$|H(j\omega)| = A/[(\cos \omega T - B) + \sin^2 \omega T]^{0.5}$$

The phase response of the filter – or to be precise the argument of $H(j\omega)$,

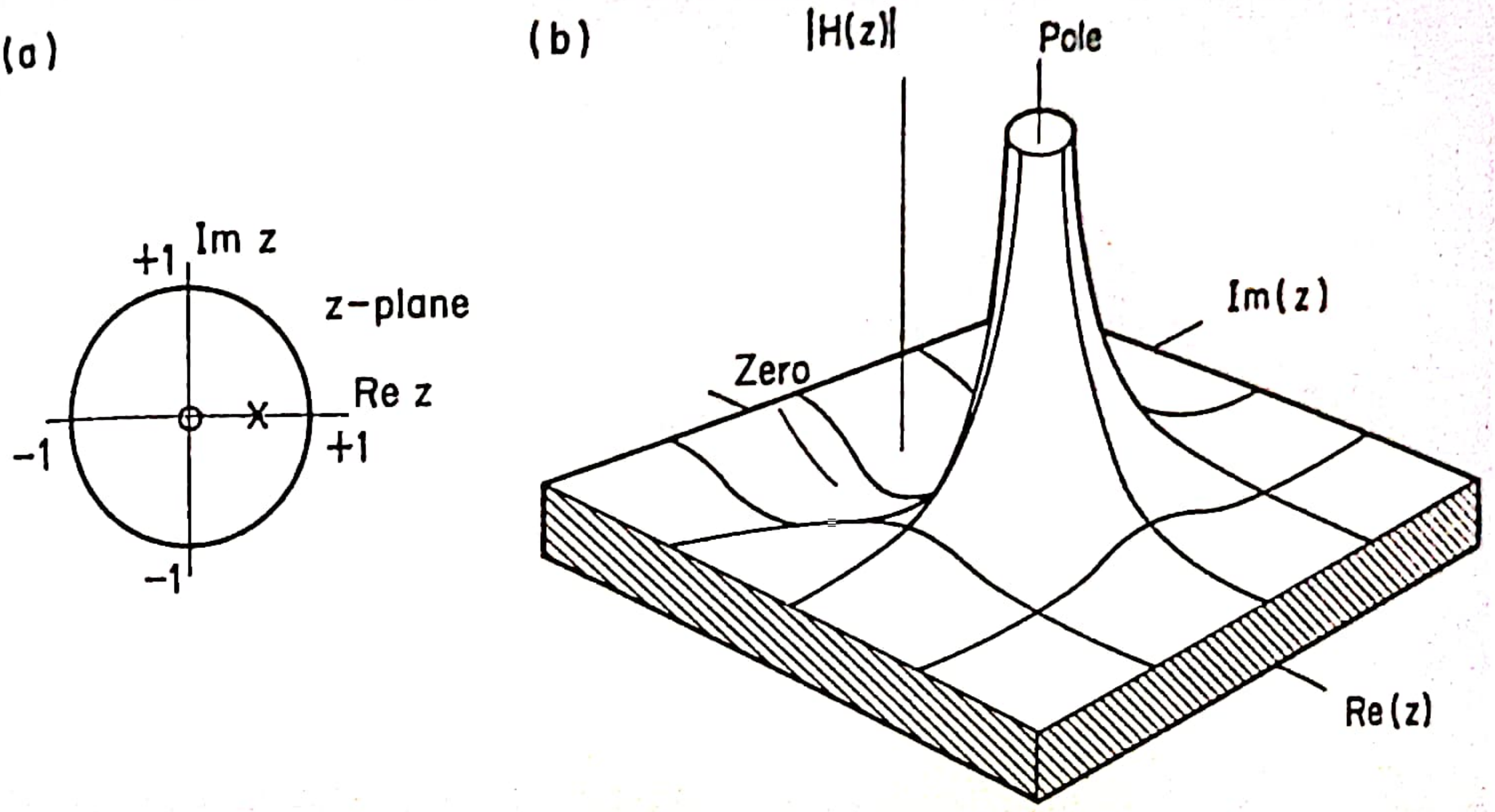


Figure 6.16 Transfer function $H(z) = Az/(z - B)$ on z-plane (a) and on 'rubbersheet' model (b)

conveniently written as $\text{Arg}[H(j\omega)]$ – is simply the angle of the numerator minus the angle of the denominator.

To use these formulae effectively, pay particular attention to the geometrical meaning of the complex number $z = e^{j\omega T} = \cos \omega T + j \sin \omega T$. Notice that the magnitude is always unity, and the angle ωT depends upon ω (T is simply the reciprocal of the sampling frequency $1/f_s$). So the locus of z will be a unit circle, centred at the origin of the z-plane. Since the highest frequency we can successfully process without aliasing is half the sampling frequency, the range of interest is usually $0 < \omega < \pi/T$, which corresponds to a semicircle. Figure 6.17 shows the magnitude and phase response of the low-pass filter described by the recurrence relationship

$$y(n) = 0.5x(n) + 0.5y(n - 1)$$

This example demonstrates how to design a digital filter using z-plane poles and zeros. The performance of the filter may then be determined geometrically, without recourse to analysis using complex numbers. The simplicity of this method makes it particularly attractive, and with practice quickly sketching the amplitude response from the pole–zero pattern becomes routine.

Consider how we might improve the low-pass characteristic of the filter considered previously. Examination of the amplitude response suggests limited attenuation around the Nyquist frequency. Since our intention is to design a low-pass filter, and the Nyquist frequency is the highest frequency we can successfully process, greater high-frequency attenuation is required. Placing a zero on the circumference of the unit circle located at $z = -1$ guarantees maximum attenuation at this frequency, thereby improving the low-pass characteristic. Figure 6.18 shows the pole–zero diagram.

To deduce the amplitude of the frequency response geometrically requires that

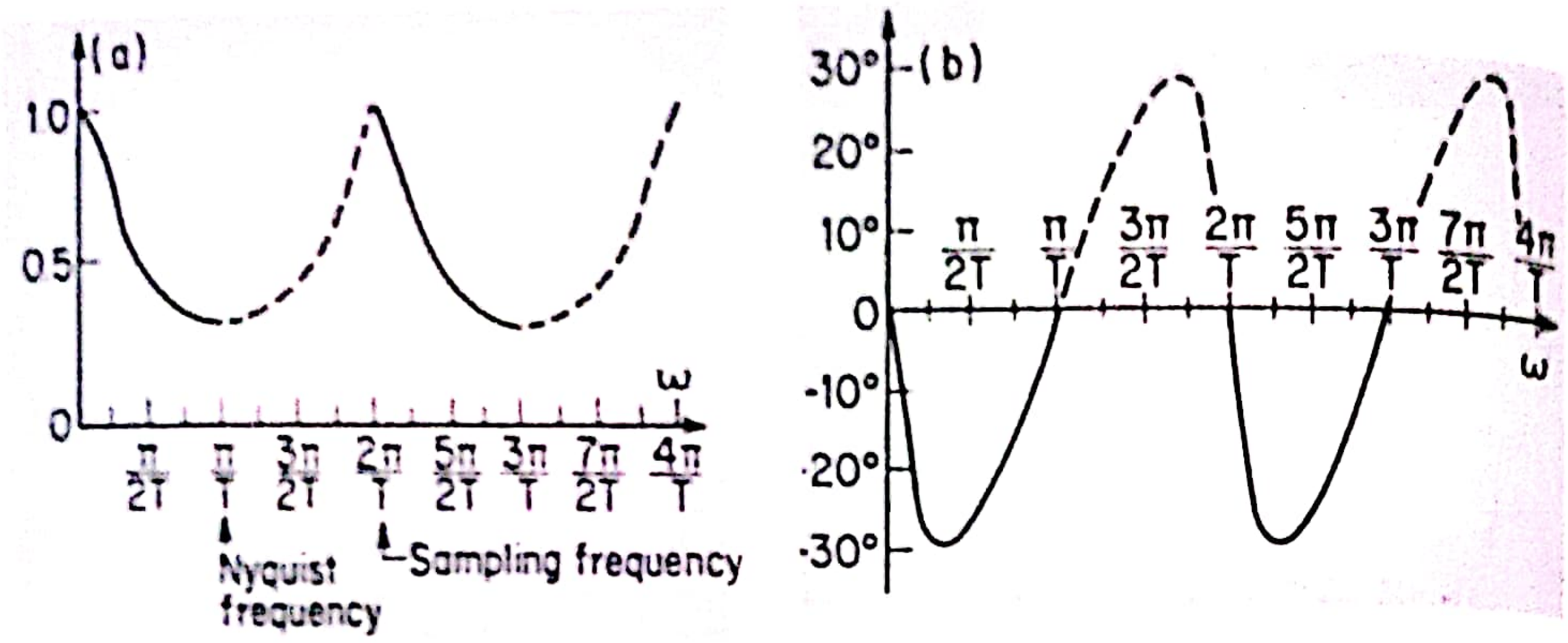


Figure 6.17 Spectrum of digital filter is repeated version of low-pass response from $\omega = 0$ to $2\pi/T$. Aliasing occurs beyond Nyquist frequency

we calculate $|H(j\omega)|$ or, in plain English, the magnitude of the zero vector divided by the magnitude of the pole vector, for values of frequency from $f = 0$ to the Nyquist frequency ($f = f_s/2$).

The resultant phase response is given by the argument of the zero vector minus the argument of the pole vector. Using these rules we can produce a reasonable amplitude and phase response of the digital filter relatively painlessly. Figure 6.19 illustrates the process.

To become proficient in this method of predicting the filter's behaviour (i.e. designing in the z-plane) try moving the position of the pole and then the zero. How does the frequency response vary? I hope you found that moving the pole along the negative real axis modified the filter's behaviour, transforming it into a high-pass characteristic. Cascading a low-pass and a high-pass filter would produce band-pass characteristics, although greater selectivity could be obtained by making the poles complex.

Band-pass digital filters

For reasons of simplicity we have deliberately avoided any reference to complex poles. We will now relax this constraint and demonstrate how greater flexibility can be incorporated into filter design, by the inclusion of complex poles. As an illustrative example we propose to design a band-pass filter with characteristics similar to those of a series resonant circuit. In keeping with earlier work, we will identify the transfer function and impulse response before writing a real-time C program. The design will intentionally use simple coefficients with a primitive microprocessor-based system in mind. A careful choice of pole–zero locations ensures simple analysis.

The digital filter will be designed on its own merits. The poles are located at half the Nyquist frequency (i.e. 0.25 times the sampling frequency) where $\omega T = \pi/2$ radians, and at a radius of 0.5.

Since the poles are complex, they will be represented as the conjugate pair $\pm j0.5$. This ensures a resonant frequency at a quarter of the sampling frequency.

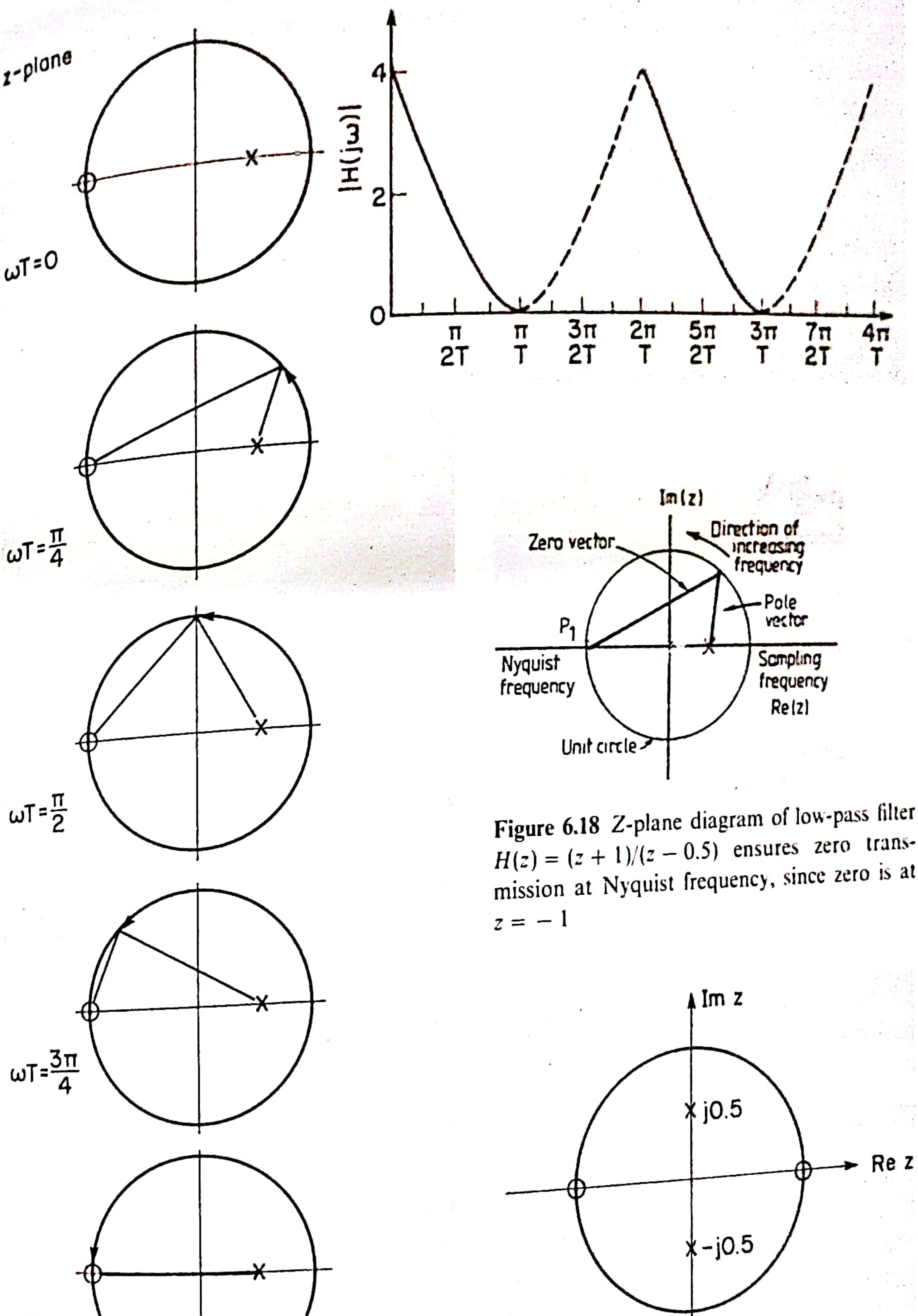


Figure 6.19 Sketching frequency response from pole/zero diagram

Figure 6.18 Z-plane diagram of low-pass filter $H(z) = (z + 1)/(z - 0.5)$ ensures zero transmission at Nyquist frequency, since zero is at $z = -1$

Figure 6.20 As an example of design with complex poles, the pole/zero diagram has simple coefficients to streamline the arithmetic

Additionally we choose to suppress the DC component by placing a zero at $z = 1$. The inclusion of a zero at $z = -1$ ensures complete attenuation of any component at the Nyquist frequency.

Referring to Figure 6.20, we may write the transfer function directly as

$$H(z) = \text{location of zeros location of poles}$$
$$= [(z + 1)(z - 1)]\,[(z + j0.5)(z - j0.5)]$$

Putting $H(z) = Y(z)\,X(z)$ we obtain the transfer function in terms of the input and output signals:

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

Cross-multiplying gives the result

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

To obtain the recurrence relationship we must convert from transforms to sequences:

$$y(n + 2) = x(n + 2) - x(n) - 0.25y(n)$$

Expressing in terms of the current output:

$$y(n) = x(n) - x(n - 2) - 0.25y(n - 2)$$

The recurrence relationship is implemented in software form by the C program Listing 6.3. It is always advisable to dry-run your programs before attempting any real-time testing, complete with A-to-D and D-to-A converters. The program shown is particularly instructive because it incorporates an impulse response test. Unit sample 1, 0, 0, 0, . . . is input term-by-term from the keyboard and the processed response displayed on the monitor. Contrast this recursive method of obtaining the impulse response with the method of convolution outlined in Chapter 5. Obviously, recursive methods are more suitable for a real-time digital signal processor.

Real-time band-pass filter

The required recurrence relationship and software structure has already been successfully tested. Reorganizing Listing 6.3 as a real-time band-pass filter in conjunction with the digital processing circuit shown in Figure 6.1 is straightforward. Listing 6.4 shows the anatomy of the program, well littered with comments.

Running the program in real-time gave a sampling rate of 250 μs and a sampling frequency of 4 kHz. Since the poles of this system are located at a quarter of the sampling frequency we must expect a maximum output or resonance in the region of 1 kHz. Figure 6.24, the photograph of the oscilloscope display, identifies the sinusoidal input to the A-to-D converter together with the processed output, which confirms the theory. Applying a square wave of 1000 Hz to the linear processor resulted in a sinusoidal output, because all the harmonics

Listing 6.3 Impulse response testing using recurrence

```
/*.....................
* SOFTWARE TESTING *
* IMPULSE RESPONSE *
.....................*/
#include <stdio.h>
main()
{
int a,b,k;
double c,d,e,f;
k = 0;
printf("Input impulse from keyboard \n");
start:scanf("%d",&a);
/*-------------
I/P UNIT IMPULSE
FROM KEYBOARD
-------------*/
d = a - c - 0.25 * f;
printf("Sample No: %d \t",k);
printf("%f \n",d);
k = k + 1;
/*-------------------------
DIFFERENCE EQUATION
y(n) = x(n) - x(n - 2) - 0.25y(n - 2)
-------------------------*/
c = b;
b = a;
f = e;
e = d;
goto start;
}
```

Listing 6.4

```
/*.....................................
* BAND-PASS DIGITAL FILTER *
* y(n) = x(n) - x(n-2) - 0.25y(n-2) *
.....................................*/
#include<stdio.h>
#include<conio.h>
#define BASE 768
main()
{
double a,b,c,d,e,f;
unsigned int y;
outp(BASE,1);
/*-------------
SELECT I/P CHANNEL
-------------*/
start:outp(BASE + 2,0);
/*-------------
START CONVERSION
-------------*/
a = 0.00392 * inp(BASE + 2);
/*-------------
NORMALISE INPUT
-------------*/
d = a - c - 0.25 * f;
y = (unsigned int)(128 + 128 * d);
/*-------------------------
MODIFY THE PROCESSED O/P:OFFSET
BINARY CODING IN D-TO-A
-------------------------*/
outp(BASE + 4,y);
c = b;
b = a;
f = e;
e = d;
/*-------------------------
DIFFERENCE EQUATION
y(n) = x(n) - x(n-2) - 0.25y(n-2)
-------------------------*/
goto start;
}
```

apart from the fundamental had been removed, so providing further evidence of the selectivity of this band-pass filter.

Modifying the program to process AC signals using the computer system shown in Figure 6.21 requires care, particularly if the A-to-D and D-to-A employ offset coding. For example, an 8-bit A-to-D conditioned to accept signals in the range -5 V to $+5$ V maps $+5$ V to 255 and -5 V to 0, while a similarly conditioned D-to-A maps 255 to $+5$ V, 128 to 0 V and 0 to -5 V. To avoid the idiosyncracies of this type of coding, an ordered approach is required. In keeping with our earlier work I shall call data captured by the A-to-D 'contents' and the data to be processed by the program $x(n)$, where $x(n)$ = contents $-$ 128. Before writing the processed output $y(n)$ to the D-to-A, the following modification is introduced:

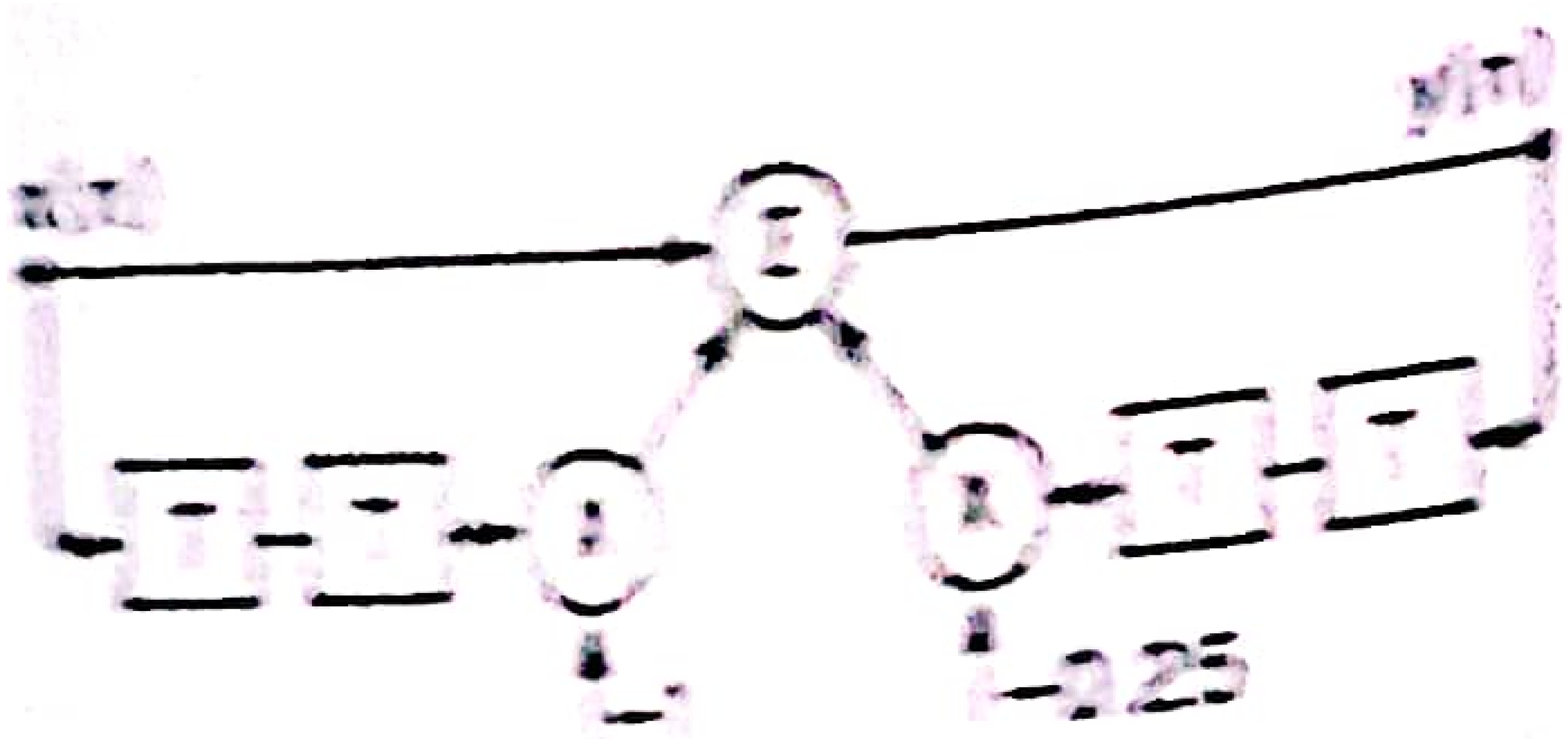


Figure 6.21 System block diagram of the band-pass filter represented by the recursive relationship $y(n) = x(n) - x(n-2) - 0.25y(n-2)$

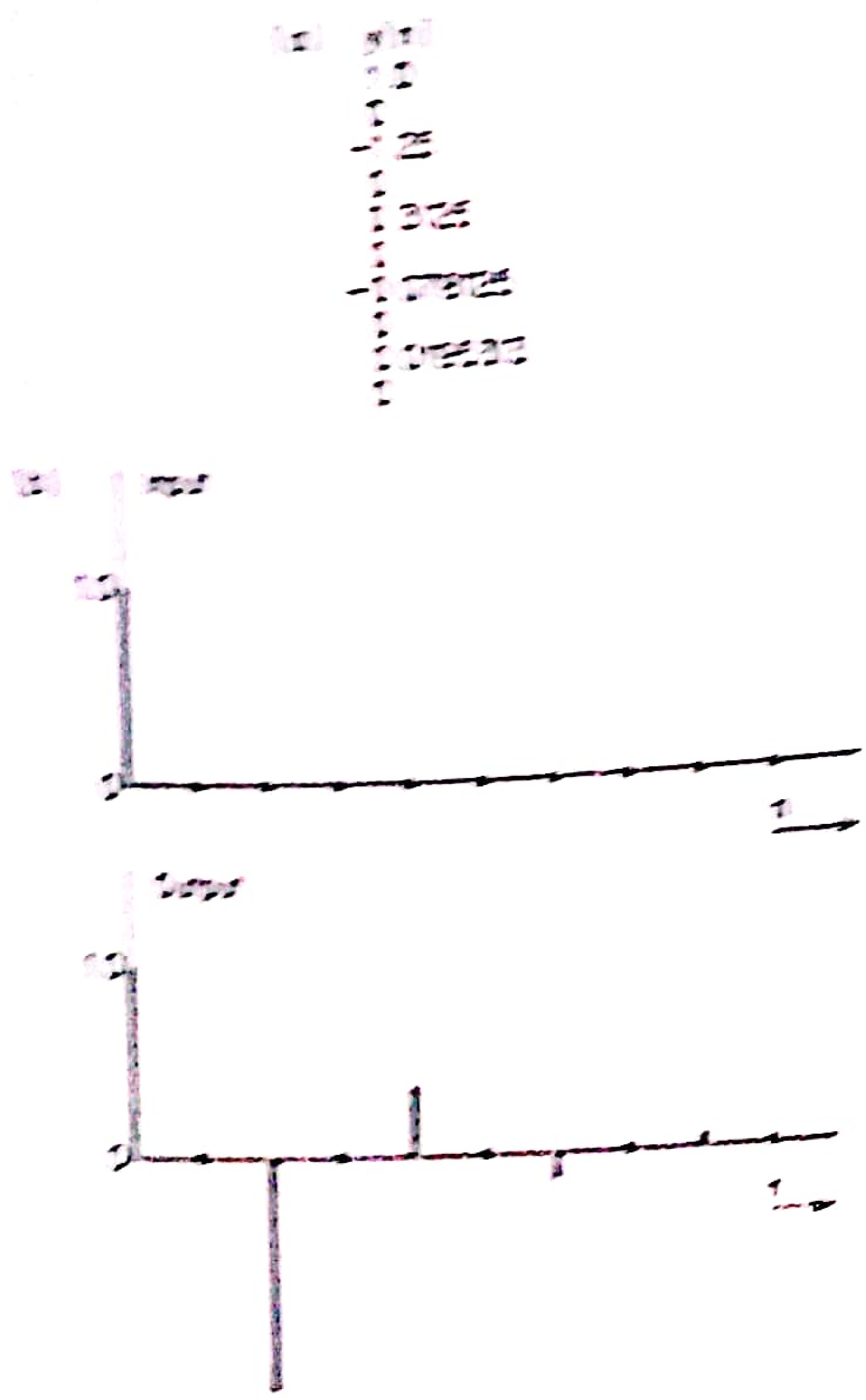


Figure 6.22 Response from impulse-response test in digital filter design program. Values are at (a) and time-domain transient response (b) oscillates at 0.25 sampling frequency

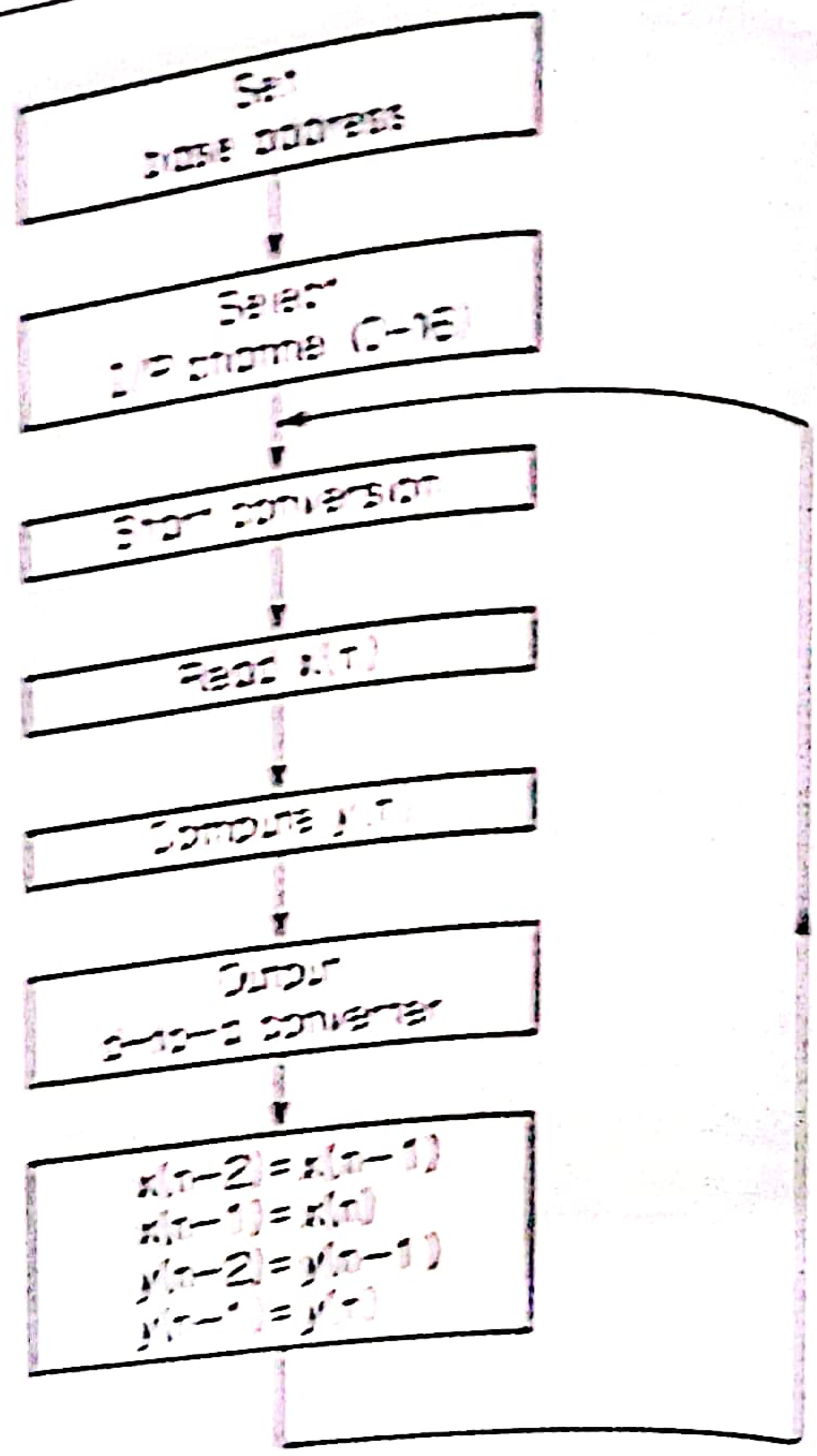


Figure 6.23 Flow chart for Listing 6.2

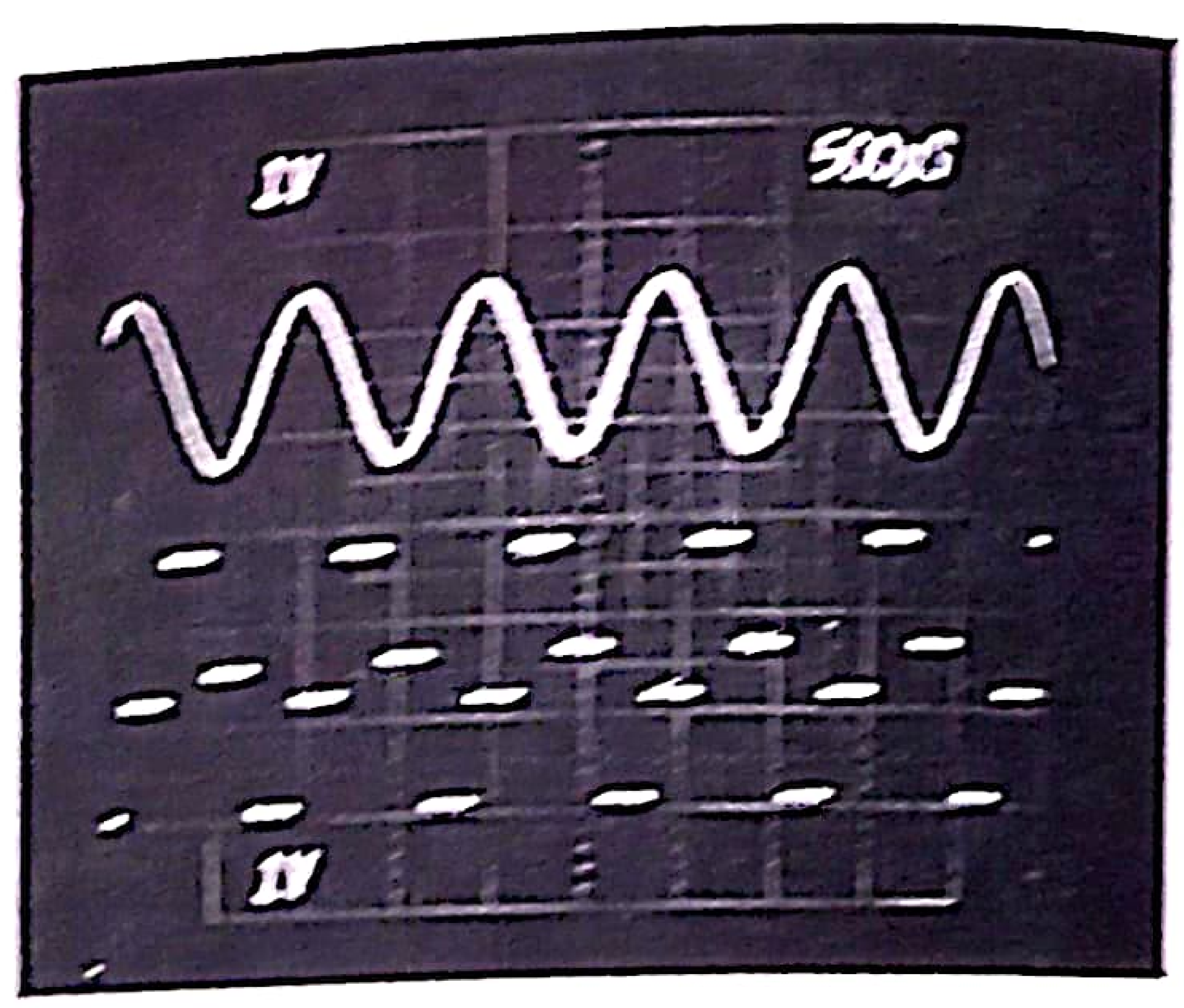


Figure 6.24 Oscilloscope display of input (1 kHz sine wave) and D-to-A output. Sampling interval is 250 μs

$$\text{output} = 127(y(n) + 1)$$

Table 6.1 is a convenient method of representing the history of the input signal as it is processed by the filter. The amplitude of the input signal was deliberately restricted, in this case to ± 3 V, to avoid generating numbers outside the dynamic range of the D-to-A (0 to 255). If you fail to recognize the significance of this, look again at the synthetically generated impulse response sequence. An input of unity

Table 6.1

n	x(n − 2)	x(n − 1)	x(n)	Constants	y(n − 2)	y(n − 1)	y(n)	Output
0	0	0	0.593	304	0	0	0.593	304
1	0	0.593	0	128	0	0.593	0	128
2	0.593	0	0	128	0.593	0	0.741	33
3	0	0	0	128	0	−0.741	0	128
4	0	0	0	128	128	0	0.185	151.71

generated an output of −1.25 in the third term of the sequence. Filters with a larger Q-factor would exaggerate this effect, requiring correspondingly greater attenuation in the amplitude of the input signal.

The bilinear transform

The elementary band-pass filter examined previously provided an introductory but limited model of s-plane design methods. The transfer function of the digital filter simply evolved from the locations of the poles and zeros. Rather than render the design in this way it may be advantageous to adopt the characteristics of well proven analogue filters such as the Butterworth or Chebychev designs, based on s-plane models.

The bilinear transform is an algebraic method which translates the s-plane characteristics of an analogue transfer function into a unique set of z-plane poles and zeros, from which we may determine the digital transfer function and difference equation. Under the transformation, the coordinates of the s-plane poles and zeros are translated on to the z-plane, preserving the desired amplitude characteristics at the expense of some frequency distortion.

The transform may be written as

$$s = (z - 1)/(z + 1)$$

which is bilinear in the sense that both the numerator and denominator are linear in z.

Pre-warping

As a prelude to any design, consider the problem of relating the cut-off frequency of the proposed digital filter to the cut-off frequency of the analogue prototype. Remember that s is a complex variable capable of characterizing any waveform; if we restrict our attention to sinusoids the analysis may be simplified by substituting $s = j\omega$, and expressing the bilinear transform as

$$j\omega = (e^{j\omega T} - 1)/(e^{j\omega T} + 1)$$

Symbolizing the cut-off frequency of the analogue prototype as ω_c and the cut-off frequency of the equivalent digital filter as Ω_c, we can establish the required relationship as follows

$$j\omega_c = \frac{\exp(j\Omega_c T/2)[\exp(j\Omega_c T/2) - \exp(-j\Omega_c T/2)]}{\exp(j\Omega_c T/2)[\exp(j\Omega_c T/2) + \exp(-j\Omega_c T/2)]}$$

$$j\omega_c = \frac{j2\sin(\Omega_c T/2)}{2\cos(\Omega_c T/2)}$$

$$\omega_c = \tan\frac{\Omega_c T}{2}$$

With knowledge of the parameters of the digital filter Ω_c and the sampling interval T, straightforward substitution produces the cut-off frequency of the analogue prototype ω_c, a technique known as pre-warping.

Expressing the transform in its inverse form

$$z = (1 - s)/(1 + s),$$

and substituting $s = j\omega$, so that

$$z = (1 - j\omega)/(1 - j\omega)$$

provides a demonstration of how the $j\omega$ axis in the s-plane is transformed on to the circumference of the unit circle in the z-plane. Figure 6.25 is the relevant diagram. If you have the time, progressively increase the angular frequency from zero towards infinity, and observe the effects of this mapping. It helps to understand the reality behind the abstraction.

The analogue Butterworth amplitude characteristic is designed to be maximally flat at low frequencies, to roll-off sharply at the cut-off frequency and to fall to zero at very high frequencies. Any digital design should follow the analogue prototype as closely as possible, emulating the flat characteristics in the pass-band, a fall in gain of -3 dB at the cut-off frequency, followed by a fall to zero at half the sampling frequency (not infinity as in the case of the analogue filter).

The magnitude-squared expression of an nth-order Butterworth filter is defined by the expression

$$|H(j\omega)|^2 = 1/[1 + (\omega/\omega_0)^{2n}]$$

Normalizing the expression by writing $\omega_0 = 1\,\text{rad}^{-1}$ leads to a more tractable expression with a -3 dB cut-off frequency of $1\,\text{rad}^{-1}$. Therefore, we may write the normalized magnitude-squared response of a second-order filter as

$$|H(j\omega)|^2 = 1/(1 + \omega^4)$$

The general form of an all-pole second-order filter is given by the transfer function

$$H(s) = 1/(as^2 + bs + c)$$

To calculate the magnitude-squared function, it is usually easier to evaluate $H(s).H(-s)$ initially, before substituting $s = j\omega$:

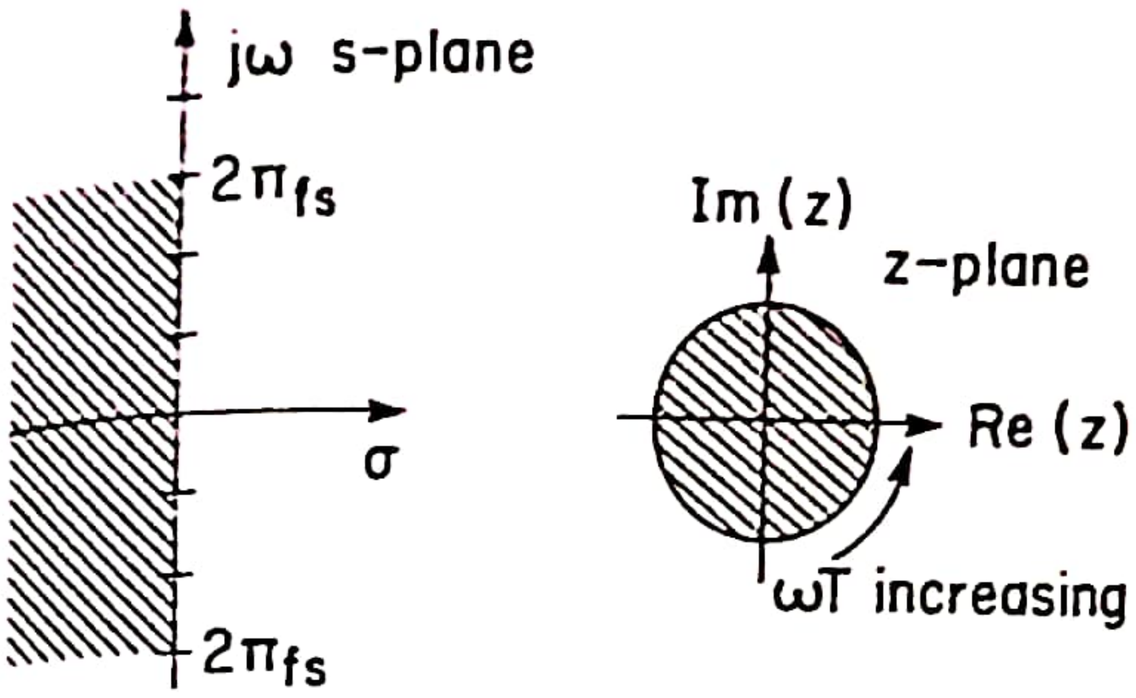


Figure 6.25 Substituting $z = e^{j\omega t}$ transforms the $j\omega$ axis of the s-plane onto circumference of unit circle in z-plane

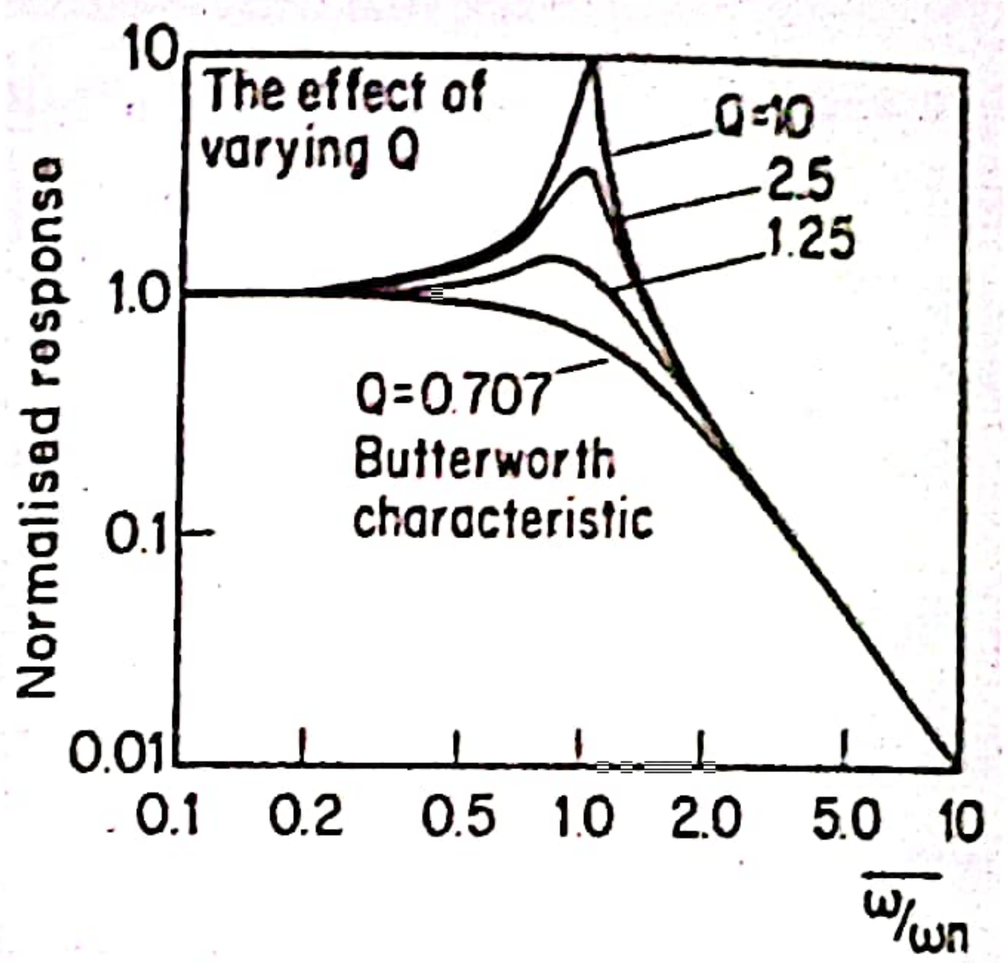


Figure 6.26 Butterworth filter designed using the bilinear transform. Curve of $Q = 1/\sqrt{2}$ gives best 'brick-wall' response

$$|H(j\omega)|^2 = H(s)H(-s),$$

where $s = j\omega$

$$H(s)H(-s) = \frac{1}{(as^2 + bs + c)} \frac{1}{(as^2 - bs + c)} = 1/[a^2s^4 + (2ac - b^2)s^2 + c^2]$$

Substituting $s = j\omega$, we obtain the frequency-domain model

$$|H(j\omega)^2| = 1/[a^2\omega^4 + (b^2 - 2ac)\omega^2 + c^2]$$

Comparing coefficients with the Butterworth magnitude response, we may write

$$a^2 = 1$$
$$b^2 - 2ac = 0$$
$$c^2 = 1$$

Hence, the second-order normalized Butterworth transfer function is given by

$$H(s) = 1/(s^2 + 1.414s + 1)$$

By inspection of the transfer function, we notice that the Q-factor is $1/\sqrt{2}$, which gives the best approximation to an ideal brick-wall response, as shown in Figure 6.26 for a variety of Q-factors.

Design example

Our intention is to design a second-order low-pass filter based on a Butterworth prototype. The digital filter will have a cut-off frequency of 1 kHz, together with a sampling frequency of 2.5 kHz.

As the complexity of the filter increases, so the time taken to complete the various additions, subtractions and multiplications escalates. Real-time signal

processing requires that all the calculations associated with each input sample are completed, before the next one becomes available. Since the software implementation of each design is likely to be unique, it is not possible to be prescriptive. The suggested sampling frequency provides a useful start for second-order Butterworth design exclusively in C.

Pre-warping, we may write the cut-off frequency of the analogue prototype as

$$\omega_c = \tan(\Omega_c T)/2$$

Substituting for the angular cut-off frequency Ω_c and sampling period T gives the numerical result

$$\omega_c = \tan[(2\pi \times 1000)/(2 \times 2500] = \tan(1.256) = 3.0777$$

Hence, we may conclude that the cut-off frequency of the analogue prototype is 3.0777 rads^{-1}.

Denormalizing, we replace s by s/ω_c in the Butterworth transfer function, modifying the analogue prototype so that

$$H(s) = 9.4722/(s^2 + 4.3518s + 9.4722)$$

To obtain the digital transfer function $H(z)$, we apply the bilinear transform, substituting $s = (z - 1)/(z + 1)$. The transfer function is

$$H(z) = \frac{9.4722}{\left[\frac{z-1}{z+1}\right]^2 + 4.3518\left[\frac{z-1}{z+1}\right] + 9.4722}$$

$$H(z) = \frac{9.4722(z+1)^2}{(z-1)^2 + 4.3518(z-1)(z+1) + 9.4722(z+1)^2}$$

$$= \frac{9.4722z^2 + 18.9444z + 9.4722}{14.842z^2 + 16.9444z + 6.1204}$$

Converting from transforms to sequences, we obtain the recurrence relationship

$$y(n) = 0.6389x(n) + 1.2779x(n-1) + 0.6389x(n-2) - 1.1430y(n-1) - 0.4129y(n-2)$$

Despite our best efforts the computational efficiency of this program could not achieve a sampling interval better than 400 μs. For the purpose of comparison, remember, the straight-wired I/O program, Listing 6.1, provided a benchmark of 25 μs.

Software-based multiplication is time critical. Disregarding any normalizing or off-set coding, the recursive structure of this filter required five multiplications per input sample. No doubt combining assembly language with C offers greater potential for throughput, but at the expense of program complexity and development time. The moral is obvious – if you want it fast, keep it simple.

The previous examples have been restricted to establishing a correspondence

Listing 6.5

```
/*****************************
* BUTTERWORTH SECOND-ORDER *
* DIGITAL FILTER FS = 2.5KHz *
* CUT-OFF FREQUENCY = 1000Hz *
*****************************/
#include <stdio.h>
#include <conio.h>
#define BASE 768
main()
{
float a,b,c,d,e,f;
/*---------------------------
NOTATION
a = x(n) b = x(n-1) c = x(n-2)
d = y(n) e = y(n-1) f = y(n-2)
---------------------------*/
unsigned int contents;
outp(BASE,1);
/*---------------
SELECT I/P CHANNEL
---------------*/
for(;;)
{
outp(BASE + 2,0);
/*-------------
START CONVERSION
-------------*/
contents = inp(BASE + 2);
a = 0.00392 * contents;
/*-------------
NORMALIZE INPUT
-------------*/
d = 0.6389 * a + 1.2779 * b + 0.6389 * c -
1.1430 * e - 0.4129 * f;
outp(BASE + 4,(int)128 * (1 + d));
/*-------------------------------
WEIGHT AND SCALE: OFFSET BINARY CODING
-------------------------------*/
c = b;
b = a;
f = e;
e = d;
/*---------
SHUFFLE DATA
---------*/
}
}
```

between analogue and digital filters. Expressed most simply the problem reduced to mapping *s*-plane poles on to the *z*-plane. We shall now concentrate on digital methods and demonstrate how it is possible to achieve results which are completely impractical using analogue circuits.

A fundamental feature of the moving averager is the ability to store input data and output it later through the D-to-A converter. This characteristic is sufficiently interesting for us to investigate it further and to produce some unusual digital signal processing effects. As we have already explained, no real analogue system can have more zeros than poles, yet the behaviour of the moving averager is characterized exclusively by zeros. If any poles are included for real-time realization, they will be located at the origin and will have no effect on the magnitude of the frequency response.

Generating echo and reverberation

Many low-pass filters exist expressly for the purpose of removing noise and improving the quality of the audio signal. Undoubtedly, all the digital filters discussed so far could be realized using combinations of R, L and C or the equivalent active filter design. But certain signal-processing operations are notoriously difficult to implement using analogue circuits: in particular, computations which require the accumulation of data over a long period.

Classical methods, that is, pre-digital, relied on analogue delay lines or tape loops to look back in time and record the signal's history. One of the fundamental advantages of digital computation is the ease with which data can be stored and

manipulated. For example, a digital signal processor can easily manipulate data captured substantially before the current sample, without any loss of accuracy.

To generate an echo, it is necessary to record or store a signal before releasing it a fraction of a second later, together with the current input. Because the poles included for real-time realization are at the origin, the filter may be characterized as a finite impulse-response (FIR) type.

As a realistic application we intend to use the personal computer system shown in Figure 6.1 to generate a scaled time delay or echo. The effect of this signal processing will be to make music sound as though it is being played in a large auditorium: the sound rebounds from wall to wall, progressively fading away into silence. Reverberation or generating multiple artificial echoes is a combination of the original input signal plus suitably scaled delays. The effect on speech is quite dramatic, making it appear as though two people were speaking simultaneously in synchronism; six-string guitars sound like twelve-string instruments and concert pianos like honky-tonks.

Initially, I shall assume the interval between samples to be 250 μs, which is a reasonable estimate in terms of the complexity of the real-time algorithm written using C. The intention is simultaneously to output the current input, together with the input signal captured ten samples previously. To avoid exceeding the dynamic range of the D-to-A, both signals will be scaled; the sum of the scaling coefficients must be less than, or equal to one. In keeping with our assertion regarding speed and simplicity, both coefficients will be 0.5. In other words, the current input will be added to the delayed input and the sum divided by two, before outputting through the D-to-A.

The scaled impulse response may be described by the sequence

$$h(n) = \{0.5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.5\}$$

Our objective is to design a computer program which will model this characteristic. Using z- transform notation, we write the transfer function directly as

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1 + z^{-10}}{2}$$

Recognizing that $X(z)$ and $Y(z)$ are the transforms of the input and output signals, we cross-multiply and obtain

$$Y(z) = X(z)(1 + z^{-10})/2$$

Converting from transforms to sequences gives the recurrence relationship

$$y(n) = 0.5x(n) + 0.5x(n - 10)$$

The pole–zero diagram, weighting function and frequency response characteristic are shown in Figure 6.29.

Observe how the magnitude of the frequency spectrum varies over the range of

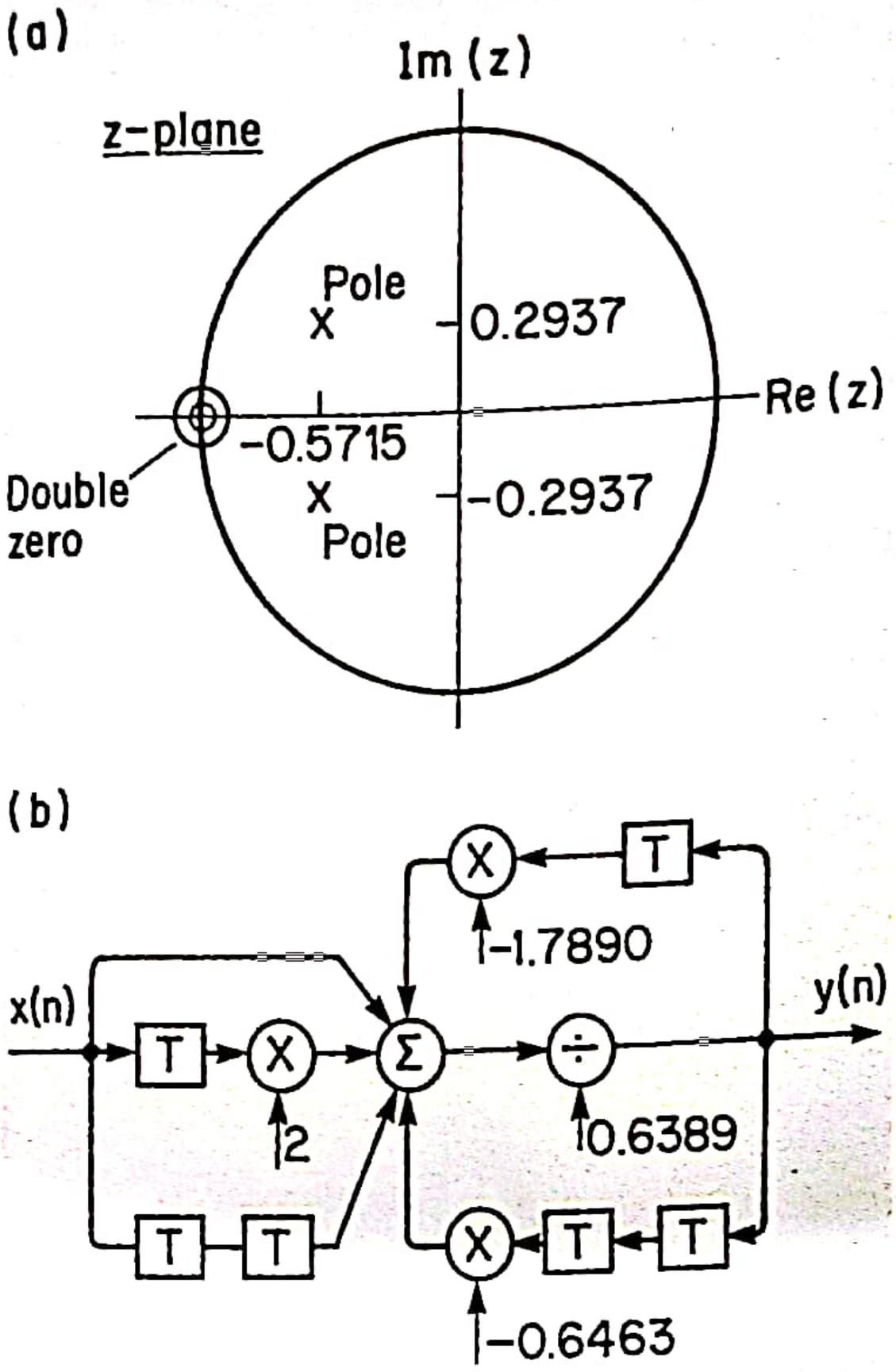


Figure 6.27 Pole/zero diagram (a) and block diagram (b) of second-order Butterworth filter. Effect of double zero at $z = -1$ in (a) reduces magnitude of frequency response to zero at Nyquist frequency. Recurrence formula indicates three forward paths in (b) and two feedback paths

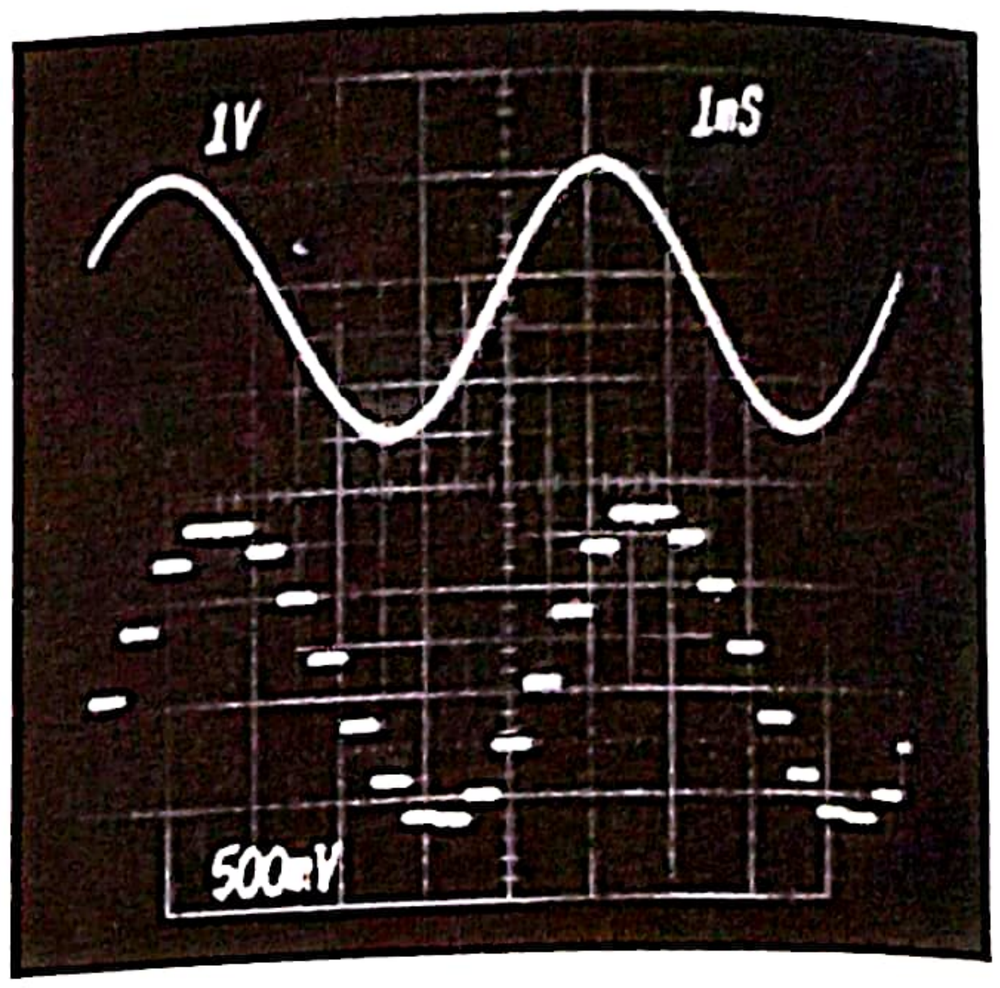


Figure 6.28 Oscilloscope display of 200 Hz (a) sine wave input and D-to-A output, after passing through Butterworth filter of Listing 6.5. Trace at (b) shows effect of double zero at half sampling frequency on 1250 Hz input

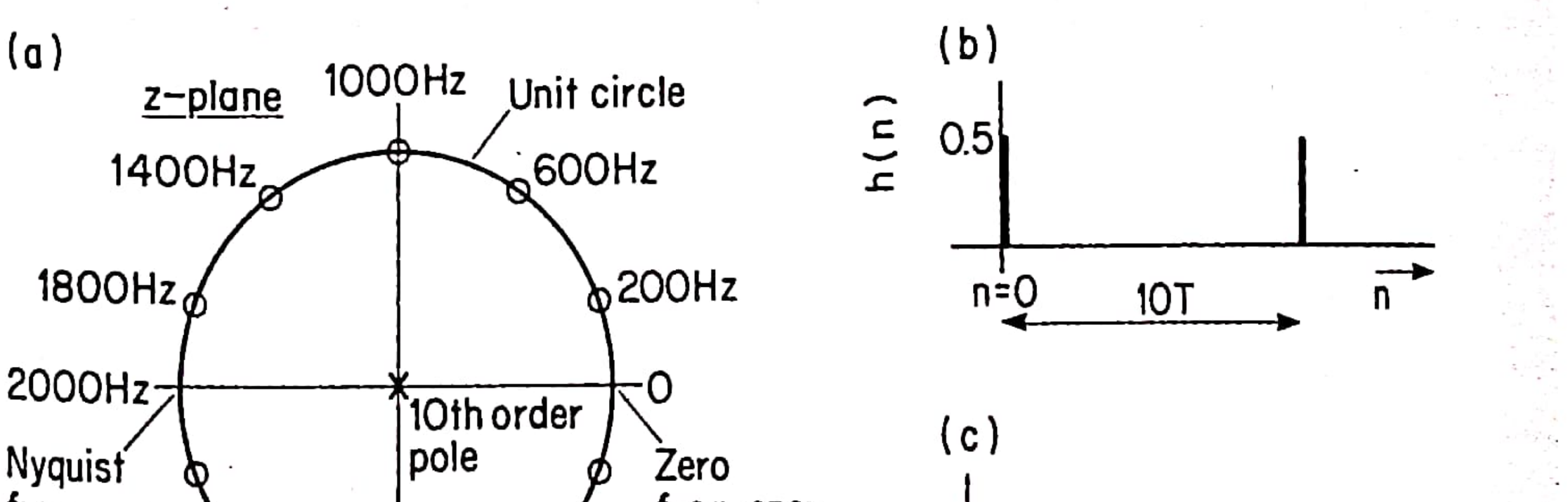


Figure 6.29 Pole/zero diagram (a), weighting function (b) and frequency spectrum resulting from all-pole echo-generating program of Listing 6.6. Frequencies whose period is multiple of twice the time delay are eliminated

Listing 6.6

```
/*****************************
* GENERATING ECHO 2.5ms DELAY *
* FS = 4000Hz *
*****************************/
#include<stdio.h>
#include<conio.h>
#define BASE 768
main()
{
float a,b,c,d,e,f,g,h,i,j,k,y;
/*_____________________
NOTATION
a = x(n) b = x(n-1) c = x(n-2)
d = x(n-3) e = x(n-4) f = x(n-5)
g = x(n-6) h = x(n-7) i = x(n-8)
j = x(n-9) k = x(n-10) y(n) = y
_____________________*/
unsigned int contents;
outp(BASE,1);
/*_____________
SELECT I/P CHANNEL
_____________*/
for(;;)
{
outp(BASE + 2,0);
/*_____________
START CONVERSION
_____________*/
contents = inp(BASE + 2);
a = 0.00392 * contents;
/*_____________
NORMALIZE INPUT
_____________*/
y = 0.5 * (a + k);
outp(BASE + 4,(int)128 * (1 + y));
k = j;
j = i;
i = h;
h = g;
g = f;
f = e;
e = d;
d = c;
c = b;
b = a;
/*_____________________
SHUFFLE DATA FOR DIFFERENCE EQN.
_____________________*/
}
}
```

interest (DC to the Nyquist frequency), producing a comb-filter characteristic. The effect of the zeros has been to completely attenuate each frequency whose period is an integral multiple of twice the time delay. Maximum output is obtained at those frequencies whose period is an integral multiple of the time delay. You can easily confirm this by running the program, Listing 6.6. On my system the sampling frequency was 4 kHz, and the designed delay 2.5 ms. Consequently, the amplitude of the filtered output fell to zero at 200, 600, 1000, 1400 and 1800 Hz.

Reverberation

Rather than using the computer and associated peripherals to generate a single delay, we choose to modify our design to generate multiple echoes or reverberation. By adding the current, non-delayed input to weighted previous inputs, the signal will cycle through the processor until it becomes too soft to be heard. Mathematically, this signal-processing operation may be represented in the time domain by the expression

$$y(t) = x(t) + ax(t - T) + a^2x(t - 2T) + \ldots a^kx(t - kT)$$

Commuting from the time-domain to the complex frequency domain using Laplace transforms, we may write

$$Y(s) = X(s) + ae^{-sT}X(s) + a^2e^{-s2T}X(s) + \ldots a^ke^{-skT}X(s)$$

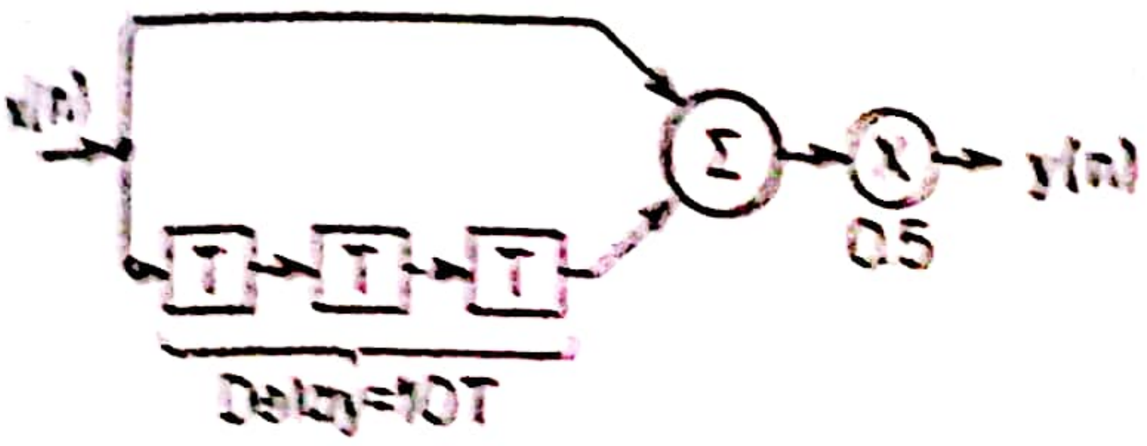


Figure 6.30 In Listing 6.6 delay elements hold input for 10 sampling periods. Delayed input is then summed with current input

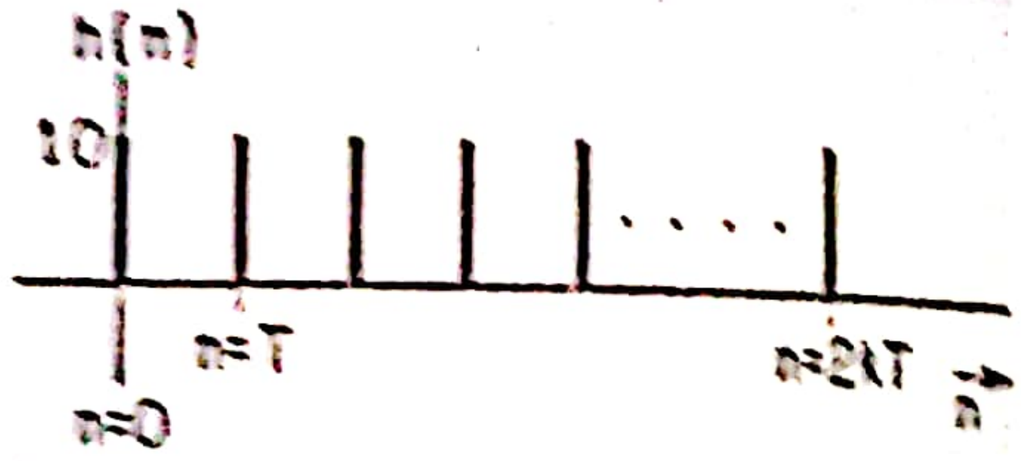
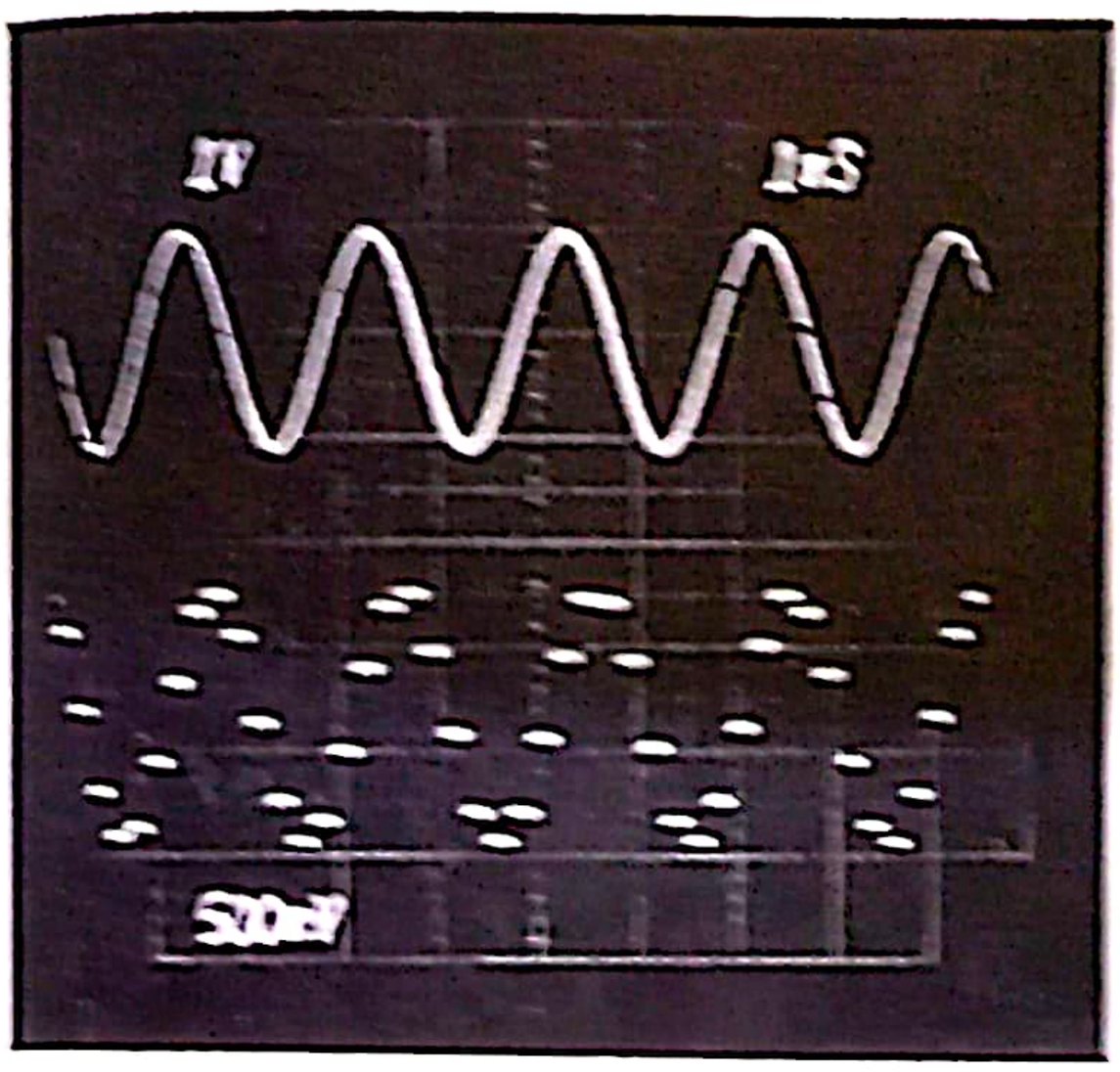


Figure 6.32 Computational efficiency is achieved by expressing the truncated impulse response (a) as the difference between the IIR (b) and weighted delayed IIR (c)



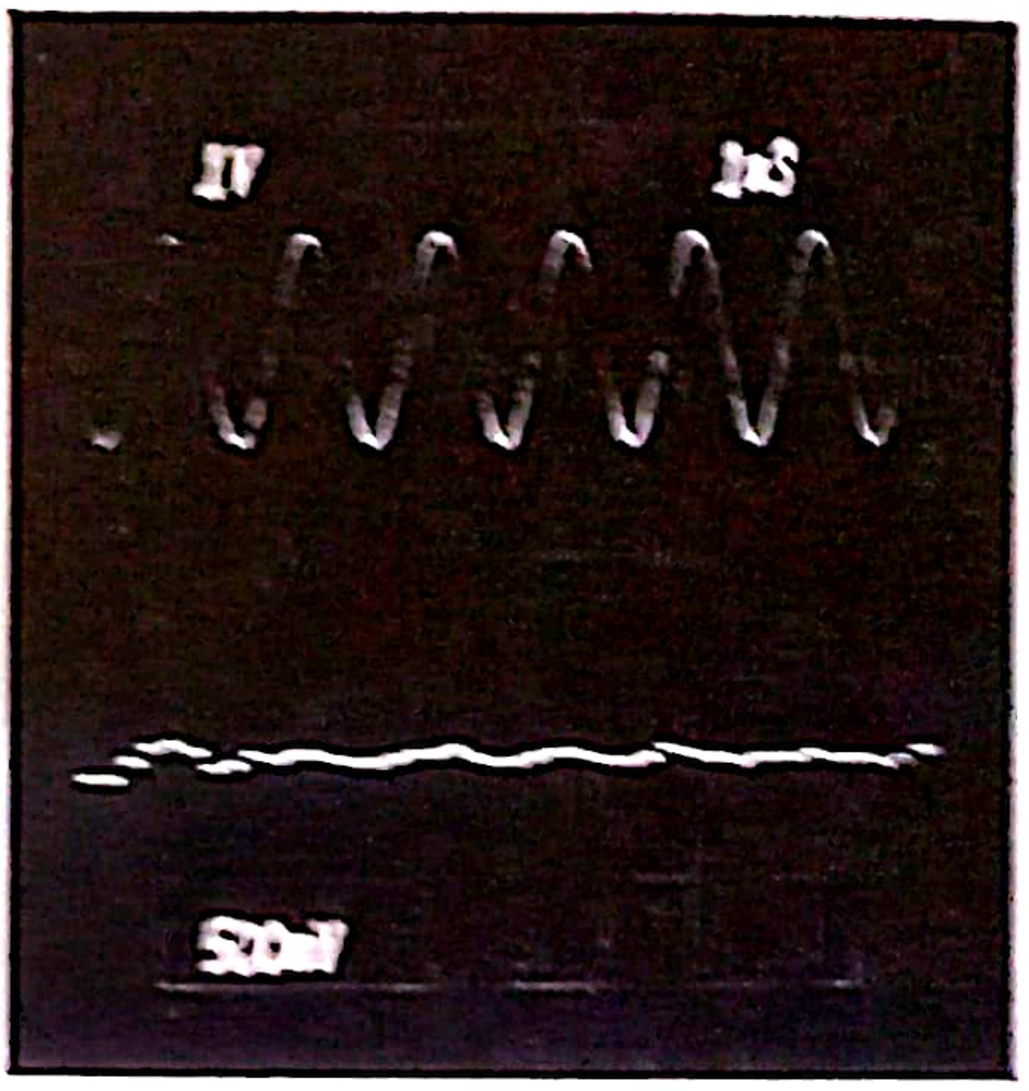


Figure 6.31 Display of 500 Hz (a) input and D-to-A output after passing through comb filter of Figure 6.29 with 2.5 ms delay. Trace at (b) shows effect of zero at $\omega T = 3\pi/10$ rad on 500 Hz input

Before expressing the signal in terms of z-transforms

$$Y(z) = X(z) + z^{-1}X(z) + z^{-2}X(z) \ldots z^{-N}X(z)$$

Inspection of the mathematical model indicates that the transfer function of the filter is a truncated version of the infinite impulse response already identified with the first-order low-pass digital filter. The required weighting function may be formed by subtracting the delayed, but weighted impulse response from the infinite impulse response as shown in Figure 6.32

$$H(z) = \frac{z}{(z-a)} - z^{-N}\frac{z}{(z-a)} = \frac{z}{(z-a)}(1 - z^{-N})$$

When a computer is programmed to behave a real-time filter, the amount of computation needed to achieve a particular impulse response is often an important consideration. Software-based multiplication is likely to impose the greatest time overhead. In the quest for greater speed, considerable computational advantage can be gained by minimising the number of multiplications, if possible by expressing the transfer function recursively. As we have already seen, this is

one of the time-critical parameters of any digital signal-processing system, because the amount of real-time processing limits the sampling frequency.

Converting from transforms to sequences, we may express the processed output $y(n)$ as

$$y(n) = ay(n-1) + x(n) - a^k x(n-k)$$

This recursive expression offers exciting possibilities for the real-time computation of certain FIR filters. Summing the current non-delayed input, with progressively scaled previous inputs, produces multiple echoes or reverberation. For example the impulse response

$$h(n) = \{1.0, 0.5, 0.25, 0.125, 0.0625\}$$

simulates the effect of adding the current input $x(n)$ to scaled versions of the four previous inputs $0.5x(n-1)$, $0.25x(n-2)$, $0.125x(n-3)$ and $0.0625x(n-4)$ respectively. Expressing the processed output recursively, we may write

$$y(n) = 0.5y(n) + x(n) - 0.03125x(n-5)$$

Recognize that only three operations are required to implement a weighting function containing five coefficients. Obviously, the greater the number of terms in the impulse response, the more efficient the recursive algorithm becomes, minimizing the number of computations to three. The pole–zero configuration, impulse response and frequency spectrum of this filter are illustrated in Figure 6.33. Observe the effect of the pole–zero cancellation (where z equals $+0.5$); this interesting result characterizes the behaviour of the filter in terms of $(k-1)$ zeros, excluding the coincident pole–zero cancellation where $z = \alpha$.

Inspection of the frequency response shows the partial amplitude attenuation

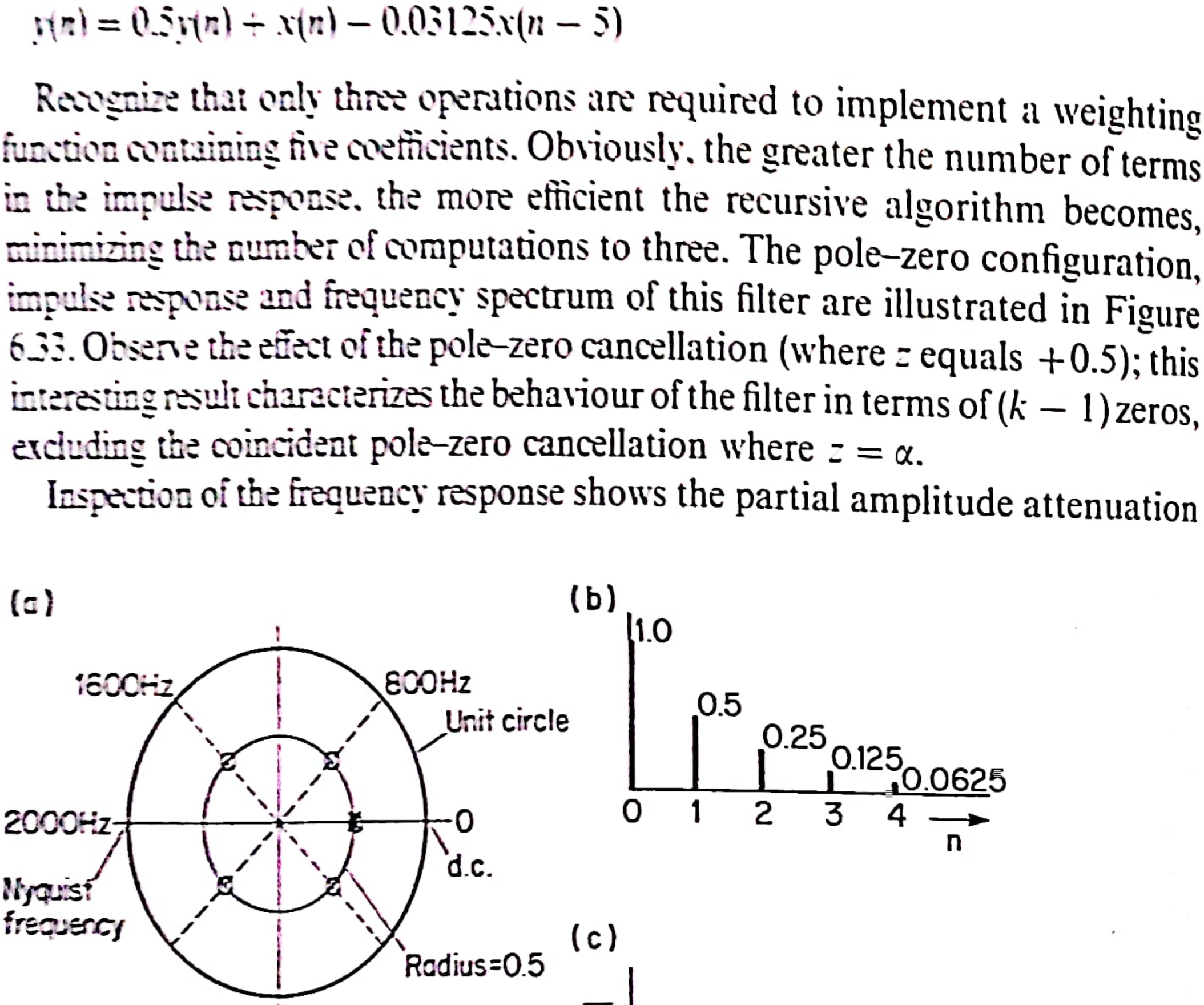
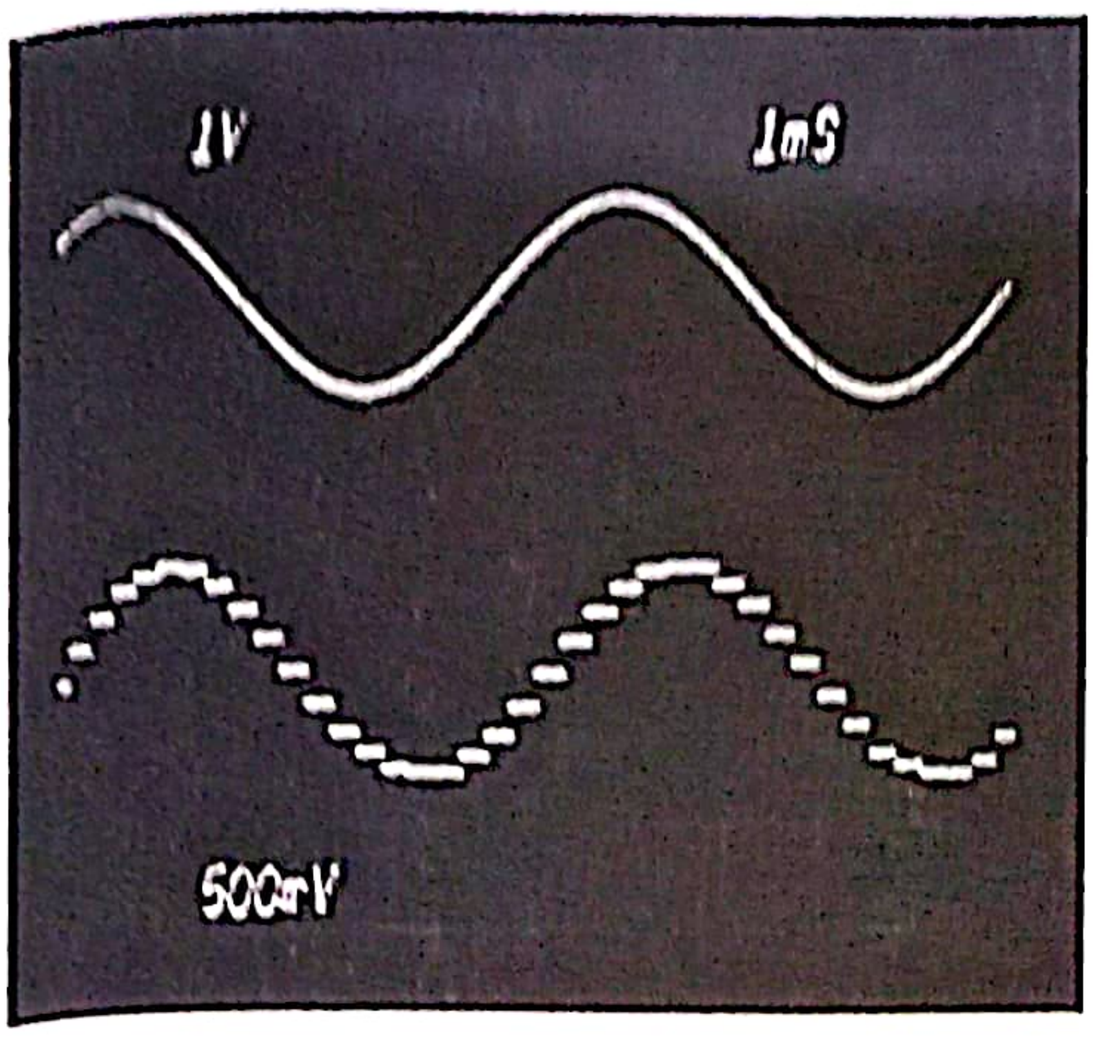


Figure 6.33 (a) Z-plane pole/zero diagram, (b) impulse response and (c) spectrum of reverberation filter of Listing 6.7



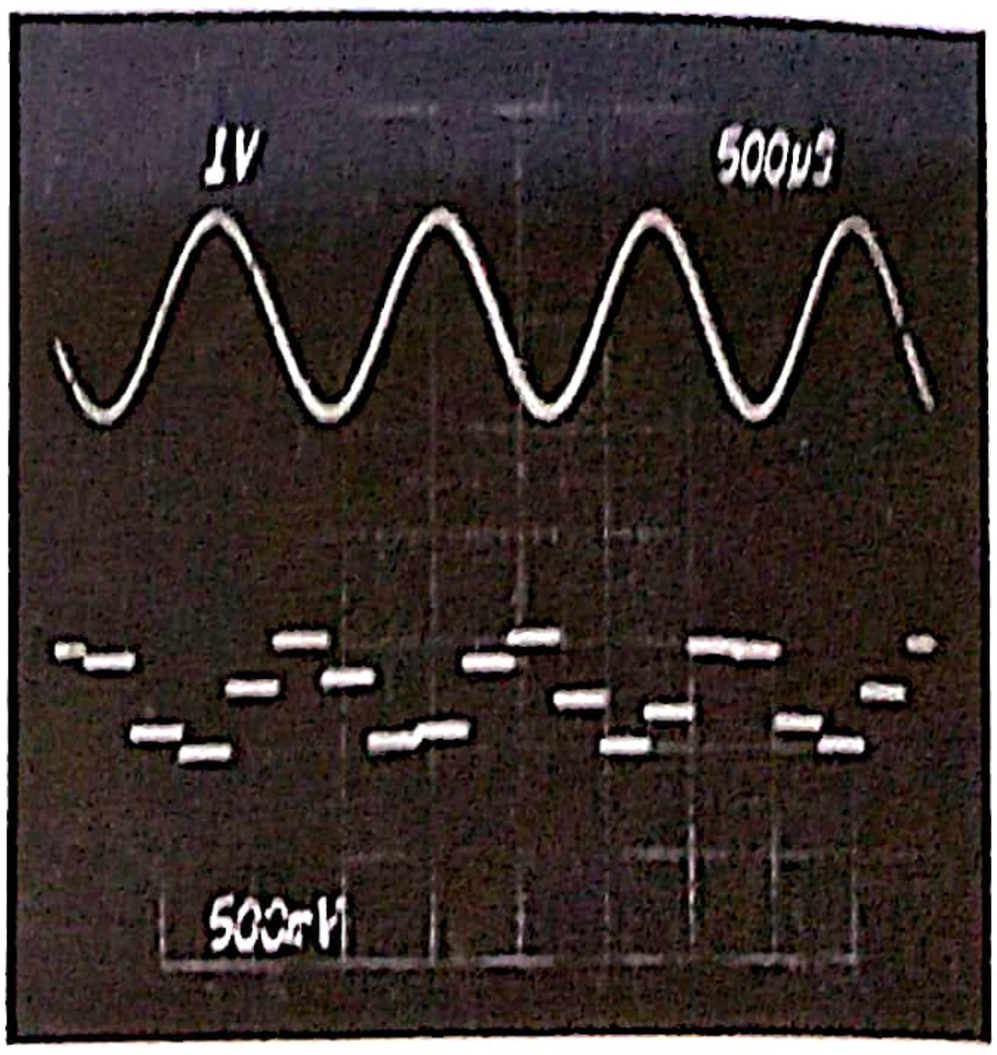


Figure 6.34 Display of 200 Hz (a) input and D-to-A output after reverberation filter of Listing 6.7. At (b), 800 Hz input shows limited attenuation of zero

Listing 6.7

```
/*************************
* GENERATING REVERBERATION *
* RECURSIVELY FS = 4000Hz *
*************************/
#include<stdio.h>
#include<conio.h>
#define BASE 768
main()
{
float a,b,c,d,e,f,y;
/*-----------------------
NOTATION
a = x(n) b = x(n-1) c = x(n-2)
d = x(n-3) e = x(n-4) f = x(n-5)
y = y(n) z = y(n-1)
-----------------------*/
unsigned int contents;
outp(BASE,1);
/*----------
SELECT I/P CHANNEL
----------*/
for(;;)
{
outp(BASE + 2,0);
/*----------
START CONVERSION
----------*/
contents = inp(BASE + 2);
a = 0.00392 * contents;
/*----------
NORMALIZE INPUT
----------*/
y = 0.5 * z + a - 0.03125 * f;
outp(BASE + 4,(int)128 * (1 + 0.516229 * y));
/*-----------------------------
WEIGHT AND SCALE: OFFSET CODING
AVOID OVERDRIVING D-TO-A(0.516229)
-----------------------------*/
f = e;
e = d;
d = c;
c = b;
b = a;
z = y;
/*-----------------------------
SHUFFLE DATA INTO RECURRENCE FORM
-----------------------*/
}
}
```

introduced by the remaining zeros. Geometrically, it is not hard to see the reason for the shape of this frequency spectrum; the magnitude is characterized completely by the radial displacement of the zeros. Remember that locating a zero on the circumference of the unit circle guarantees complete attenuation at that frequency, as previously demonstrated by the digital delay/comb filter. As usual, the poles at the origin make no contribution to the shape of the spectrum amplitude.

Further reading

Bissell, C.C. (1987) A phasor approach to digital filters. *Int. J. Engineering Education*, **24**

Bogner, R.E. and Constantinides, A.G. (1975) *Introduction to Digital Filtering*. John Wiley

Chamberlin, H. (1977) A sampling of techniques for computer performance of music. *Byte*, September

Cheetham, B.M.G. and Hughes, P. (1982) Digital filter design. *Wireless World*, May.

Hutchings, H.J. (1985) Digital filters explained. *Wireless World*, December.

O'Haver, T. (1978) Audio processing with a microprocessor. *Byte*, June.

Sylvester Bradley, J.T.R. (1983) Digital filter design. *Wireless World*, May.

Electronic Signal Processing (1984) T 326. Open University Press.