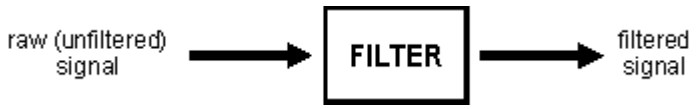
INTRODUCTION TO DIGITAL FILTERS

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 equalisers in hi-fi systems, and many other areas.

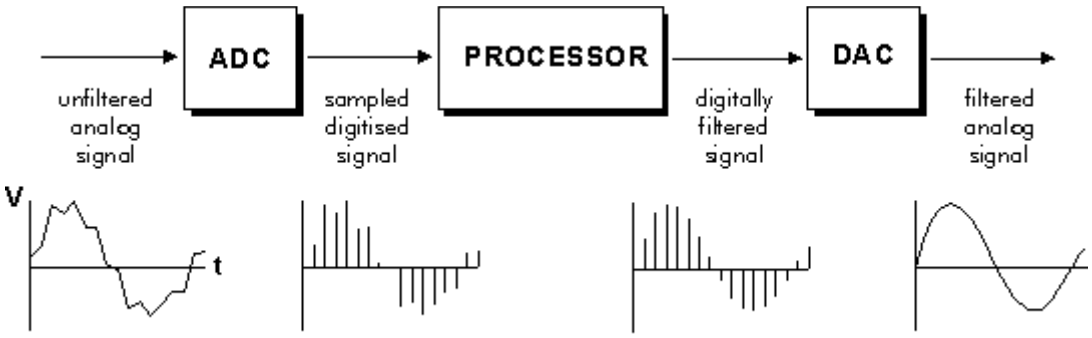
There are well-established standard techniques for designing an analog filter circuit for a given requirement. At all stages, the signal being filtered is an electrical voltage or current which is the direct analogue of the physical quantity (e.g. a sound or video signal or transducer output) involved.

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 specialised DSP (Digital Signal Processor) chip.

The analog input signal must first be sampled and digitised 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.

Note that in a digital filter, the signal is represented by a sequence of numbers, rather than a voltage or current.

The following diagram shows the basic setup of such a system.



Advantages of using digital filters

The following list gives some of the main advantages of digital over analog filters.

1. A digital filter is *programmable*, i.e. its operation is determined by a program stored in the processor's memory. This means the digital filter can easily be changed without affecting the circuitry (hardware). An analog filter can only be changed by redesigning the filter circuit.

2. Digital filters are easily *designed*, *tested* and *implemented* on a general-purpose computer or workstation.

3. The characteristics of analog filter circuits (particularly those containing active components) are subject to drift and are dependent on temperature. Digital filters do not suffer from these problems, and so are extremely *stable* with respect both to time and temperature.

4. Unlike their analog counterparts, digital filters can handle *low frequency* signals accurately. As the speed of DSP technology continues to increase, digital filters are being applied to high frequency signals in the RF (radio frequency) domain, which in the past was the exclusive preserve of analog technology.

5. Digital filters are very much more *versatile* in their ability to process signals in a variety of ways; this includes the ability of some types of digital filter to adapt to changes in the characteristics of the signal.

6. Fast DSP processors can handle complex combinations of filters in parallel or cascade (series), making the hardware requirements relatively *simple* and *compact* in comparison with the equivalent analog circuitry.

Operation of digital filters

In this section, we will develop the basic theory of the operation of digital filters. This is essential to an understanding of how digital filters are designed and used.

Suppose the "raw" signal which is to be digitally filtered is in the form of a voltage waveform described by the function

$$V = x(t)$$

where t is time.

This signal is sampled at time intervals h (the sampling interval). The sampled value at time $t = ih$ is

$$x_i = x(ih)$$

Thus the digital values transferred from the ADC to the processor can be represented by the sequence

$$x_0, x_1, x_2, x_3, \ldots$$

corresponding to the values of the signal waveform at

$$t = 0, h, 2h, 3h, \ldots$$

and $t = 0$ is the instant at which sampling begins.

At time $t = nh$ (where n is some positive integer), the values available to the processor, stored in memory, are

$$x_0, x_1, x_2, x_3, \ldots x_n$$

Note that the sampled values x_{n+1}, x_{n+2} etc. are not available, as they haven't happened yet!

The digital output from the processor to the DAC consists of the sequence of values

$$y_0, y_1, y_2, y_3, \ldots y_n$$

In general, the value of y_n is calculated from the values $x_0, x_1, x_2, x_3, \ldots, x_n$. The way in which the y's are calculated from the x's determines the filtering action of the digital filter.

In the next section, we will look at some examples of simple digital filters.

Examples of simple digital filters

The following examples illustrate the essential features of digital filters.

1. Unity gain filter:

$$y_n = x_n$$

Each output value y_n is exactly the same as the corresponding input value x_n:

$$y_0 = x_0$$
$$y_1 = x_1$$
$$y_2 = x_2$$
$$\ldots etc$$

This is a trivial case in which the filter has no effect on the signal.

2. Simple gain filter:

$$y_n = Kx_n$$

where K = constant.

This simply applies a gain factor K to each input value.

$K > 1$ makes the filter an amplifier, while $0 < K < 1$ makes it an attenuator. $K < 0$ corresponds to an inverting amplifier. Example (1) above is simply the special case where $K = 1$.

3. Pure delay filter:

$$y_n = x_{n-1}$$

The output value at time $t = nh$ is simply the input at time $t = (n-1)h$, i.e. the signal is delayed by time h:

$$y_0 = x_{-1}$$
$$y_1 = x_0$$
$$y_2 = x_1$$
$$y_3 = x_2$$
$$\ldots \text{ etc}$$

Note that as sampling is assumed to commence at $t = 0$, the input value x_{-1} at $t = -h$ is undefined. It is usual to take this (and any other values of x prior to $t = 0$) as zero.

4. Two-term difference filter:

$$y_n = x_n - x_{n-1}$$

The output value at $t = nh$ is equal to the difference between the current input x_n and the previous input x_{n-1}:

$$y_0 = x_0 - x_{-1}$$
$$y_1 = x_1 - x_0$$
$$y_2 = x_2 - x_1$$
$$y_3 = x_3 - x_2$$
$$\ldots \text{ etc}$$

i.e. the output is the *change* in the input over the most recent sampling interval h. The effect of this filter is similar to that of an analog differentiator circuit.

5. Two-term average filter:

$$y_n = \frac{x_n + x_{n-1}}{2}$$

The output is the average (arithmetic mean) of the current and previous input:

$$y_0 = \frac{x_0 + x_{-1}}{2}$$
$$y_1 = \frac{x_1 + x_0}{2}$$
$$y_2 = \frac{x_2 + x_1}{2}$$
$$y_3 = \frac{x_3 + x_2}{2}$$
… etc

This is a simple type of low pass filter as it tends to smooth out high-frequency variations in a signal. (We will look at more effective low pass filter designs later).

6. Three-term average filter:

$$y_n = \frac{x_n + x_{n-1} + x_{n-2}}{3}$$

This is similar to the previous example, with the average being taken of the current and two previous inputs:

$$y_0 = \frac{x_0 + x_{-1} + x_{-2}}{3}$$
$$y_1 = \frac{x_1 + x_0 + x_{-1}}{3}$$
$$y_2 = \frac{x_2 + x_1 + x_0}{3}$$
$$y_3 = \frac{x_3 + x_2 + x_1}{3}$$

As before, x_{-1} and x_{-2} are taken to be zero.

7. Central difference filter:

$$y_n = \frac{x_n - x_{n-2}}{2}$$

This is similar in its effect to example (4). The output is equal to half the change in the input signal over the previous two sampling intervals:

$$y_0 = \frac{x_0 - x_{-2}}{2}$$

$$y_1 = \frac{x_1 - x_{-1}}{2}$$

$$y_2 = \frac{x_2 - x_0}{2}$$

$$y_3 = \frac{x_3 - x_1}{2}$$

… etc

Order of a digital filter

The *order* of a digital filter is the number of *previous* inputs (stored in the processor's memory) used to calculate the current output.

Thus:

1. Examples (1) and (2) above are zero-order filters, as the current output y_n depends only on the current input x_n and not on any previous inputs.
2. Examples (3), (4) and (5) are all of first order, as one previous input (x_{n-1}) is required to calculate y_n. (Note that the filter of example (3) is classed as first-order because it uses one previous input, even though the current input is not used).
3. In examples (6) and (7), two previous inputs (x_{n-1} and x_{n-2}) are needed, so these are second-order filters.

Filters may be of any order from zero upwards.

Digital filter coefficients

All of the digital filter examples given above can be written in the following general forms:

Zero order: $y_n = a_0 x_n$

First order: $y_n = a_0 x_n + a_1 x_{n-1}$

Second order: $y_n = a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2}$

Similar expressions can be developed for filters of any order.

The constants a_0, a_1, a_2, ... appearing in these expressions are called the *filter coefficients*. It is the values of these coefficients that determine the characteristics of a particular filter.

The following table gives the values of the coefficients of each of the filters given as examples above.

Example	Order	a_0	a_1	a_2
1	0	1	-	-
2	0	K	-	-
3	1	0	1	-
4	1	1	-1	-
5	1	$^1/_2$	$^1/_2$	-
6	2	$^1/_3$	$^1/_3$	$^1/_3$
7	2	$^1/_2$	0	$-^1/_2$

SAQ 1 For each of the following filters, state the *order* of the filter and identify the values of its coefficients:

(a) $y_n = 2x_n - x_{n-1}$

(b) $y_n = x_{n-2}$

(c) $y_n = x_n - 2x_{n-1} + 2x_{n-2} + x_{n-3}$

Recursive and non-recursive filters

For all the examples of digital filters discussed so far, the current output (y_n) is calculated solely from the current and previous input values (x_n, x_{n-1}, x_{n-2}, ...). This type of filter is said to be *non-recursive*.

A *recursive* filter is one which in addition to input values also uses previous *output* values. These, like the previous input values, are stored in the processor's memory.

The word *recursive* literally means "running back", and refers to the fact that previously-calculated output values go back into the calculation of the latest output. The expression for a recursive filter therefore contains not only terms involving the input values (x_n, x_{n-1}, x_{n-2}, ...) but also terms in y_{n-1}, $y_{n-2, \ldots}$

From this explanation, it might seem as though recursive filters require more calculations to be performed, since there are previous output terms in the filter expression as well as input terms. In fact, the reverse is usually the case: to achieve a given frequency response characteristic using a recursive filter generally requires a much lower order filter (and therefore fewer terms to be evaluated by the processor) than the equivalent non-recursive filter. This will be demonstrated later.

Note

Some people prefer an alternative terminology in which a non-recursive filter is known as an FIR (or Finite Impulse Response) filter, and a recursive filter as an IIR (or Infinite Impulse Response) filter.

These terms refer to the differing "impulse responses" of the two types of filter. The impulse response of a digital filter is the output sequence from the filter when a *unit impulse* is applied at its input. (A unit impulse is a very simple input sequence consisting of a single value of 1 at time $t = 0$, followed by zeros at all subsequent sampling instants).

An FIR filter is one whose impulse response is of finite duration. An IIR filter is one whose impulse response theoretically continues for ever because the recursive (previous output) terms feed back energy into the filter input and keep it going. The term IIR is not very accurate because the actual impulse responses of nearly all IIR filters reduce virtually to zero in a finite time. Nevertheless, these two terms are widely used.

Example of a recursive filter

A simple example of a recursive digital filter is given by

$$y_n = x_n + y_{n-1}$$

In other words, this filter determines the current output (y_n) by adding the current input (x_n) to the previous output (y_{n-1}):

$$\begin{aligned} y_0 &= x_0 + y_{-1} \\ y_1 &= x_1 + y_0 \\ y_2 &= x_2 + y_1 \\ y_3 &= x_3 + y_2 \\ &\ldots \text{ etc} \end{aligned}$$

Note that y_{-1} (like x_{-1}) is undefined, and is usually taken to be zero.

Let us consider the effect of this filter in more detail. If in each of the above expressions we substitute for y_{n-1} the value given by the previous expression, we get the following:

$$\begin{aligned} y_0 &= x_0 + y_{-1} = x_0 \\ y_1 &= x_1 + y_0 = x_1 + x_0 \\ y_2 &= x_2 + y_1 = x_2 + x_1 + x_0 \\ y_3 &= x_3 + y_2 = x_3 + x_2 + x_1 + x_0 \\ &\ldots \text{ etc} \end{aligned}$$

Thus we can see that y_n, the output at $t = nh$, is equal to the sum of the current input x_n and all the previous inputs. This filter therefore sums or *integrates* the input values, and so has a similar effect to an analog integrator circuit.

This example demonstrates an important and useful feature of recursive filters: the economy with which the output values are calculated, as compared with the equivalent non-recursive filter. In this example, each output is determined simply by adding two numbers together. For instance, to calculate the output at time $t = 10h$, the recursive filter uses the expression

$$y_{10} = x_{10} + y_9$$

To achieve the same effect with a non-recursive filter (i.e. without using previous output values stored in memory) would entail using the expression

$$y_{10} = x_{10} + x_9 + x_8 + x_7 + x_6 + x_5 + x_4 + x_3 + x_2 + x_1 + x_0$$

This would necessitate many more addition operations as well as the storage of many more values in memory.

Order of a recursive (IIR) digital filter

The *order* of a digital filter was defined earlier as the number of previous inputs which have to be stored in order to generate a given output. This definition is appropriate for non-recursive (FIR) filters, which use only the current and previous inputs to compute the current output. In the case of recursive filters, the definition can be extended as follows:

> **The order of a recursive filter is the largest number of previous input *or* output values required to compute the current output.**

This definition can be regarded as being quite general: it applies both to FIR and IIR filters.

For example, the recursive filter discussed above, given by the expression

$$y_n = x_n + y_{n-1}$$

is classed as being of first order, because it uses one previous output value (y_{n-1}), even though no previous inputs are required.

In practice, recursive filters usually require the *same number of previous inputs and outputs*. Thus, a first-order recursive filter generally requires one previous input (x_{n-1}) and one previous output (y_{n-1}), while a second-order recursive filter makes use of two previous inputs (x_{n-1} and x_{n-2}) and two previous outputs (y_{n-1} and y_{n-2}); and so on, for higher orders.

Note that a recursive (IIR) filter must, by definition, be of at least first order; a zero-order recursive filter is an impossibility. (Why?)

SAQ 2 State the order of each of the following recursive filters:

(a) $y_n = 2x_n - x_{n-1} + y_{n-1}$

(b) $y_n = x_{n-1} - x_{n-3} - 2y_{n-1}$

(c) $y_n = x_n + 2x_{n-1} + x_{n-2} - 2y_{n-1} + y_{n-2}$

Coefficients of recursive (IIR) digital filters

From the above discussion, we can see that a recursive filter is basically like a non-recursive filter, with the addition of extra terms involving previous inputs (y_{n-1}, y_{n-2} etc.).

A first-order recursive filter can be written in the general form

$$y_n = \frac{(a_0 x_n + a_1 x_{n-1} - b_1 y_{n-1})}{b_0}$$

Note the minus sign in front of the "recursive" term $b_1 y_{n-1}$, and the factor $(1/b_0)$ applied to all the coefficients. The reason for expressing the filter in this way is that it allows us to rewrite the expression in the following symmetrical form:

$$b_0 y_n + b_1 y_{n-1} = a_0 x_n + a_1 x_{n-1}$$

In the case of a second-order filter, the general form is

$$y_n = \frac{a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2} - b_1 y_{n-1} - b_2 y_{n-2}}{b_0}$$

The alternative "symmetrical" form of this expression is

$$b_0 y_n + b_1 y_{n-1} + b_2 y_{n-2} = a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2}$$

Note the convention that the coefficients of the inputs (the *x*'s) are denoted by *a*'s, while the coefficients of the outputs (the *y*'s) are denoted by *b*'s.

SAQ 3 Identify the values of the filter coefficients for the first-order recursive filter

$$y_n = x_n + y_{n-1}$$

discussed earlier.

Repeat this for each of the filters in SAQ 2.

The transfer function of a digital filter

In the last section, we used two different ways of expressing the action of a digital filter: a form giving the output y_n directly, and a "symmetrical" form with all the output terms on one side and all the input terms on the other.

In this section, we introduce what is called the *transfer function* of a digital filter. This is obtained from the symmetrical form of the filter expression, and it allows us to describe a filter by means of a convenient, compact expression. We can also use the transfer function of a filter to work out its frequency response.

First of all, we must introduce the *delay operator*, denoted by the symbol z^{-1}.

When applied to a sequence of digital values, this operator gives the *previous* value in the sequence. It therefore in effect introduces a delay of one sampling interval.

Applying the operator z^{-1} to an input value (say x_n) gives the previous input (x_{n-1}):

$$z^{-1} x_n = x_{n-1}$$

Suppose we have an input sequence

$$x_0 = 5$$
$$x_1 = -2$$
$$x_2 = 0$$
$$x_3 = 7$$
$$x_4 = 10$$

Then

$$z^{-1} x_1 = x_0 = 5$$
$$z^{-1} x_2 = x_1 = -2$$
$$z^{-1} x_3 = x_2 = 0$$

and so on. Note that $z^{-1} x_0$ would be x_{-1}, which is unknown (and usually taken to be zero, as we have already seen).

Similarly, applying the z^{-1} operator to an output gives the previous output:

$$z^{-1} y_n = y_{n-1}$$

Applying the delay operator z^{-1} twice produces a delay of two sampling intervals:

$$z^{-1} (z^{-1} x_n) = z^{-1} x_{n-1} = x_{n-2}$$

We adopt the (fairly logical) convention

$$z^{-1} z^{-1} = z^{-2}$$

i.e. the operator z^{-2} represents a delay of two sampling intervals:

$$z^{-2}\, x_n = x_{n-2}$$

This notation can be extended to delays of three or more sampling intervals, the appropriate power of z^{-1} being used.

Let us now use this notation in the description of a recursive digital filter. Consider, for example, a general second-order filter, given in its symmetrical form by the expression

$$b_0 y_n + b_1 y_{n-1} + b_2 y_{n-2} = a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2}$$

We will make use of the following identities:

$$y_{n-1} = z^{-1}\, y_n$$
$$y_{n-2} = z^{-2}\, y_n$$
$$x_{n-1} = z^{-1}\, x_n$$
$$x_{n-2} = z^{-2}\, x_n$$

Substituting these expressions into the digital filter gives

$$(b_0 + b_1 z^{-1} + b_2 z^{-2})\, y_n = (a_0 + a_1 z^{-1} + a_2 z^{-2})\, x_n$$

Rearranging this to give a direct relationship between the output and input for the filter, we get

$$\frac{y_n}{x_n} = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{b_0 + b_1 z^{-1} + b_2 z^{-2}}$$

This is the general form of the transfer function for a second-order recursive (IIR) filter.

For a first-order filter, the terms in z^{-2} are omitted. For filters of order higher than 2, further terms involving higher powers of z^{-1} are added to both the numerator and denominator of the transfer function.

A non-recursive (FIR) filter has a simpler transfer function which does not contain any denominator terms. The coefficient b_0 is usually taken to be equal to 1, and all the other b coefficients are zero. The transfer function of a second-order FIR filter can therefore be expressed in the general form

$$\frac{y_n}{x_n} = a_0 + a_1 z^{-1} + a_2 z^{-2}$$

For example, the three-term average filter, defined by the expression

$$y_n = \frac{x_n + x_{n-1} + x_{n-2}}{3}$$

can be written using the z^{-1} operator notation as

$$y_n = \frac{x_n + z^{-1}x_n + z^{-2}x_n}{3} = \frac{(1 + z^{-1} + z^{-2})\,x_n}{3}$$

The transfer function for the filter is therefore

$$\frac{y_n}{x_n} = \frac{1 + z^{-1} + z^{-2}}{3}$$

The general form of the transfer function for a first-order *recursive* filter can be written

$$\frac{y_n}{x_n} = \frac{a_0 + a_1 z^{-1}}{b_0 + b_1 z^{-1}}$$

Consider, for example, the simple first-order recursive filter

$$y_n = x_n + y_{n-1}$$

which we discussed earlier. To derive the transfer function for this filter, we rewrite the filter expression using the z^{-1} operator:

$$(1 - z^{-1})\,y_n = x_n$$

Rearranging gives the filter transfer function as

$$\frac{y_n}{x_n} = \frac{1}{1 - z^{-1}}$$

As a further example, consider the second-order IIR filter

$$y_n = x_n + 2x_{n-1} + x_{n-2} - 2y_{n-1} + y_{n-2}$$

Collecting output terms on the left and input terms on the right to give the "symmetrical" form of the filter expression, we get

$$y_n + 2y_{n-1} - y_{n-2} = x_n + 2x_{n-1} + x_{n-2}$$

Expressing this in terms of the z^{-1} operator gives

$$(1 + 2z^{-1} - z^{-2})\,y_n = (1 + 2z^{-1} + z^{-2})\,x_n$$

and so the transfer function is

$$\frac{y_n}{x_n} = \frac{1 + 2z^{-1} + z^{-2}}{1 + 2z^{-1} - z^{-2}}$$

SAQ 4 Derive the transfer functions of each of the filters in SAQ 2.

Tutorial question

A digital filter is described by the expression

$$y_n = 2x_n - x_{n-1} + 0.8y_{n-1}$$

(a) State whether the filter is recursive or non-recursive. Justify your answer.

(b) State the order of the filter.

(c) Derive the filter transfer function.

(d) The following sequence of input values is applied to the filter.

$$x_0 = 5$$
$$x_1 = 16$$
$$x_2 = 8$$
$$x_3 = -3$$
$$x_4 = 0$$
$$x_5 = 2$$

Determine the output sequence for the filter, from y_0 to y_5.

Answers to Self-Assessment Questions

SAQ 1

a) Order = 1: $a_0 = 2$, $a_1 = -1$

b) Order = 2: $a_0 = 0$, $a_1 = 0$, $a_2 = 1$

c) Order = 3: $a_0 = 1$, $a_1 = -2$, $a_2 = 2$, $a_3 = 1$

SAQ 2

a) Order = 1

b) Order = 3

c) Order = 2

SAQ 3

$a_0 = 1$ $a_1 = 0$

$b_0 = 1$ $b_1 = -1$

For filters listed in SAQ 2:

a) $a_0 = 2$ $a_1 = -1$ $b_0 = 1$ $b_1 = -1$

b) $a_0 = 0$ $a_1 = 1$ $a_2 = 0$ $a_3 = -1$ $b_0 = 1$ $b_1 = 2$ $b_2 = 0$ $b_3 = 0$

c) $a_0 = 1$ $a_1 = 2$ $a_2 = 1$ $b_0 = 1$ $b_1 = 2$ $b_2 = -1$

SAQ 4

a) $$\frac{y_n}{x_n} = \frac{2 - z^{-1}}{1 - z^{-1}}$$

b) $$\frac{y_n}{x_n} = \frac{z^{-1} - z^{-3}}{1 + 2z^{-1}}$$

c) $$\frac{y_n}{x_n} = \frac{1 + 2z^{-1} + z^{-2}}{1 + 2z^{-1} - z^{-2}}$$