DSP Configurations

In this lecture we discuss the different physical (or software) configurations that can be used to actually realize or implement DSP functions. Recall that the general form of a DSP function is a ratio of polynomials in z^{-1}. For example when we used Matlab to design a Butterworth filter, the program returned an 'a' vector and a 'b' vector,

When we executed the Matlab instruction >>[b,a] = butter(4,wn), Matlab responded with:

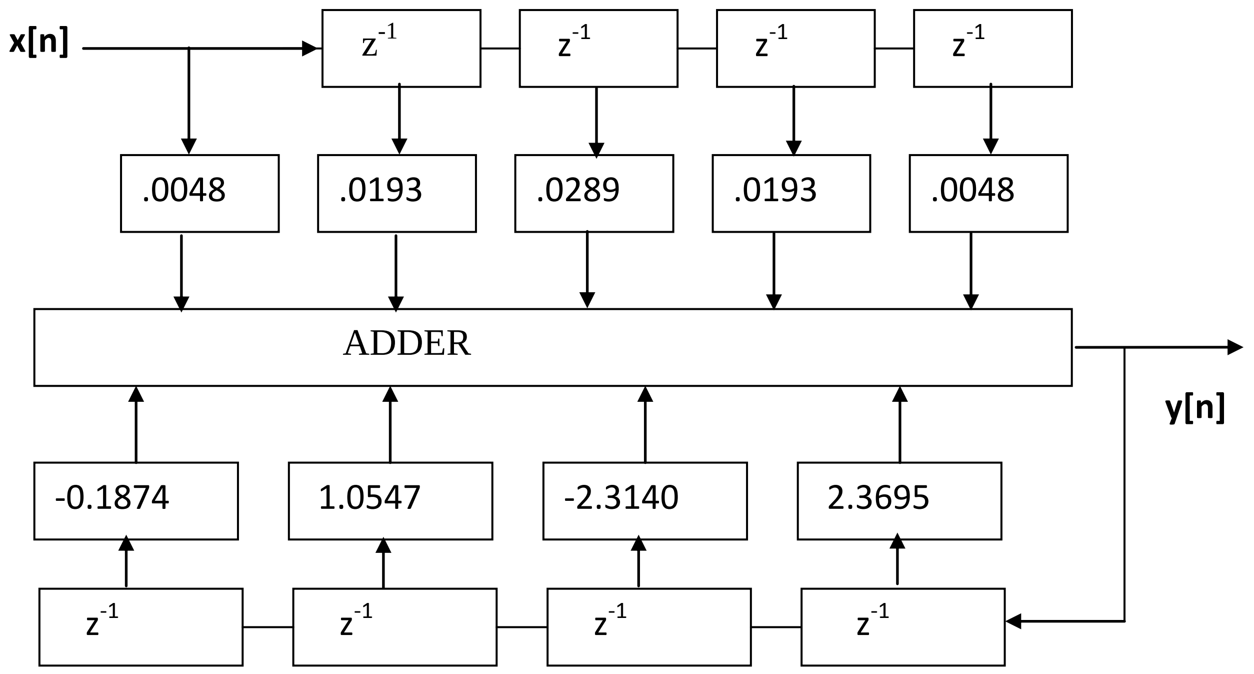
```
b =   0.0048   0.0193   0. 0289   0.0193   0.0048

a =   1.0000  -2.3695   2.3140  -1.0547   0.1874
```

thus the system function for this filter would be

$$H(z) = \frac{0.0048z^{-0} + 0.193z^{-1} + 0.0289z^{-2} + 0.0193z^{-3} + 0.0048z^{-4}}{1z^{-0} - 2.3695z^{-1} + 2.3140z^{-2} - 1.0547z^{-3} + 0.1874z^{-4}}$$

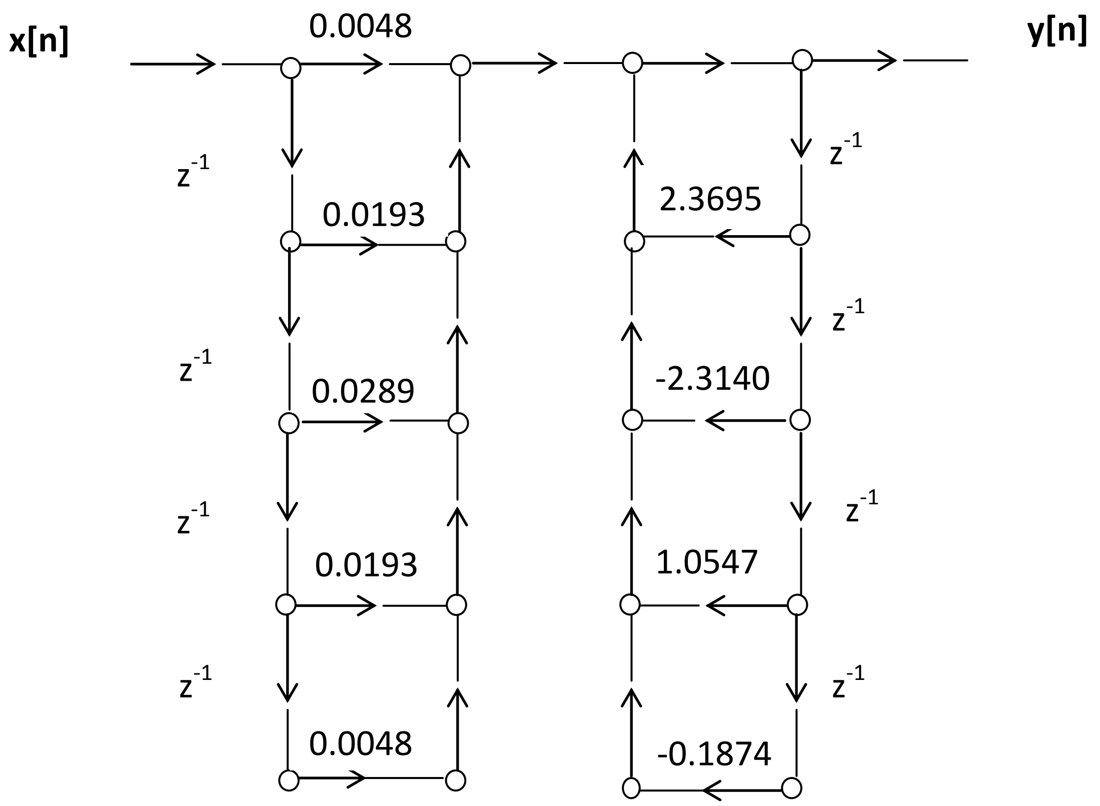
(Note the symmetry of the numerator. We will come back to this later.)

We found that we could actually implement this filter, either in hardware or software, with the following block diagram, recalling that the forward terms constitute the b vector and the feedback terms are made up of (1- the denominator terms) and z^{-1} represents one unit delay.



Note that the string of delay operations resembles a tapped delay line, leading to the description of the complexity of the circuit in terms of the number of taps.

A simpler form which accomplishes the same function is the flow diagram. The ***direct form I*** flow diagram for this filter is

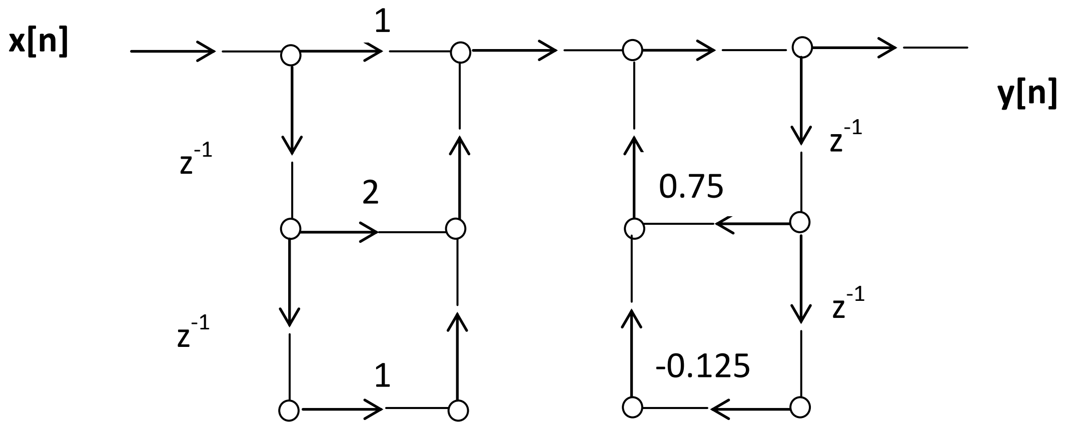


From this form we can easily calculate the number of arithmetic operations – additions, multiplies and delay operations. Each branch with an arrow and a number causes a multiplication (even if the number is 1), each node with two arrows converging requires an addition, and each z^{-1} requires a delay element.

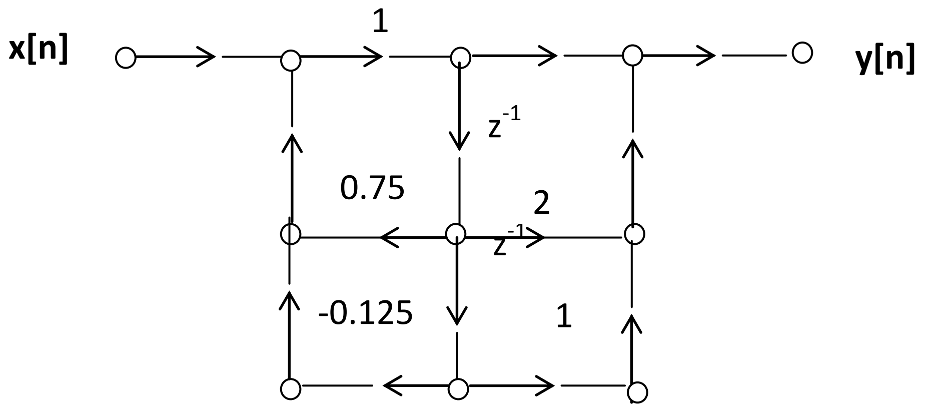
The form above results directly from the system function H(z). However, the system function can be rearranged a number of ways, by suitable factoring of the numerator and denominator and rearranging the factors. To illustrate, let us consider a simpler system function, borrowed from a textbook, which has been chosen because it lends itself to factoring and rearranging.

Consider $H(z) = \dfrac{1 + 2z^{-1} + z^{-2}}{1 - 0.75z^{-1} + 0.125z^{-2}}$

The ***direct form I*** realization of this H(z) is



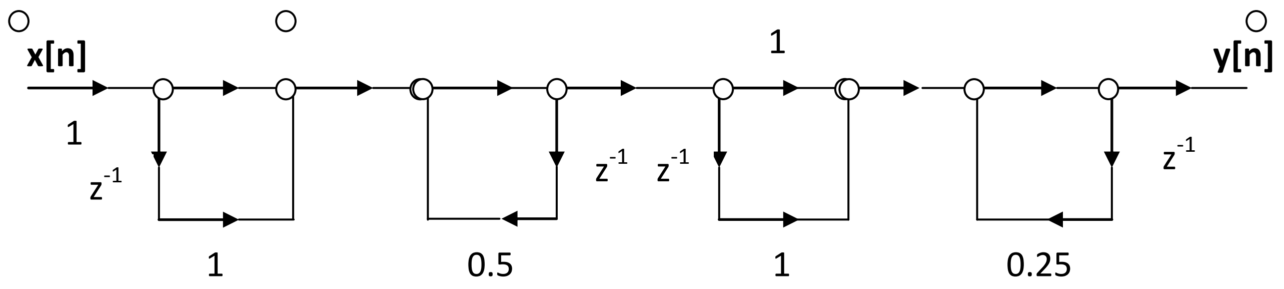
The two sections can be interchanged, thus making it possible to combine the delay elements, thereby reducing the number of delay elements (essentially storage locations). This is the ***direct form II.***



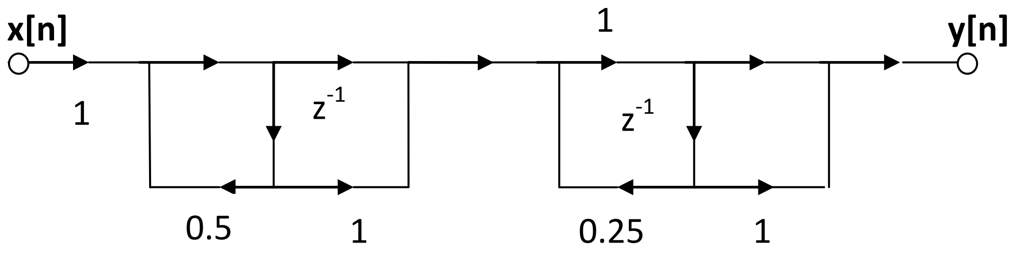
We can factor the numerator and denominator of $H(z) = \dfrac{1 + 2z^{-1} + z^{-2}}{1 - 0.75z^{-1} + 0.125z^{-2}}$

Getting $H(z) = \dfrac{1 + 2z^{-1} + z^{-2}}{1 - 0.75z^{-1} + 0.125z^{-2}} = \dfrac{(1+ z^{-1})(1+z^{-1})}{(1 - 0.5z^{-1})(1-0.25z^{-1})} = \dfrac{(1 + z^{-1})}{(1-0.5z^{-1})}\,\dfrac{(1 + z^{-1})}{(1 - 0.25z^{-1})}$

Which leads to the form:



This is called a cascade form, and as before, the forward and reverse paths can be interchanged, combining the delay elements, leading to:

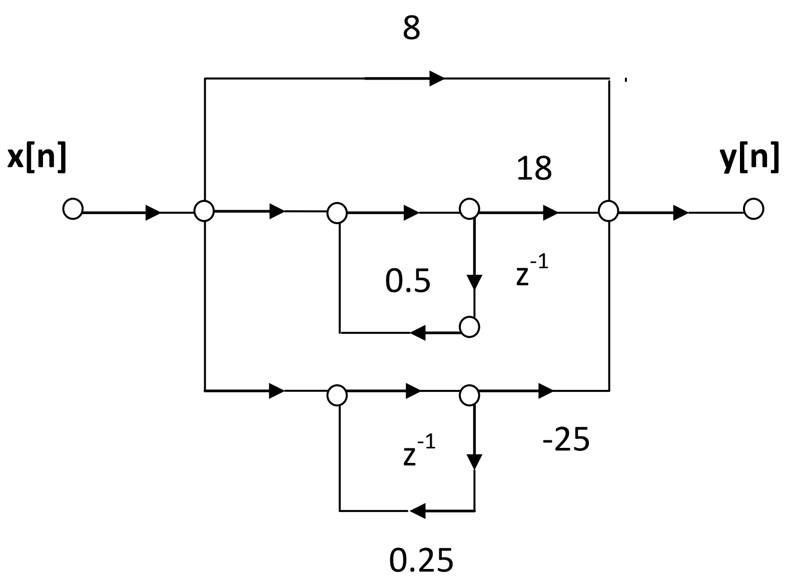


Note that the sections in the two forms above could be interchanged to form even more variations.

Another form can be found by factoring the polynomials, using partial fractions, getting:

$$H(z) = 8 + \frac{18}{1-0.5Z^{-1}} - \frac{25}{1-0.25z^{-1}}$$

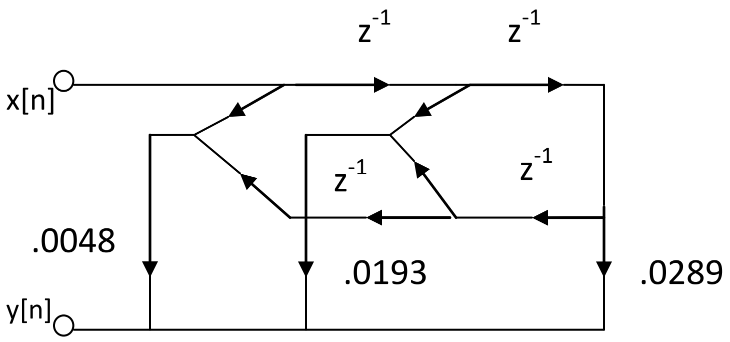
This form can be realized by the following ***parallel*** structure



<u>FIR systems</u>. Recall that for FIR systems, the denominator term is just 'a' = 1. Thus we have only the numerator to synthesize.

Let us assume we have a system function consisting only of the numerator of the filter we used earlier. Then $H(z) = 0.0048z^{-0} + 0.193z^{-1} + 0.0289z^{-2} + 0.0193z^{-3} + 0.0048z^{-4}$. Note the symmetry of the coefficients.

We can synthesize this function using the following flow diagram:



With this form, we need four delay elements, but only 3 multiply and 2 add operations, or (N/2 + 1) multiply and N/ 2 adds for a fourth order response. This helps to compensate for the fact that the **FIR** structures generally require more stages for a given filter response than the **IIR.**

We have discussed various ways in which the system functions can be broken down and rearranged. Let us now look at reasons for choosing certain forms.

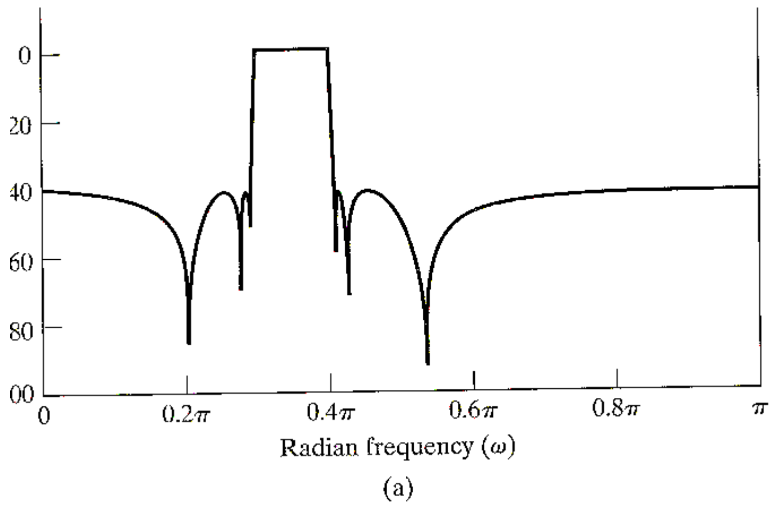
One of the major reasons for choosing different arrangements of the system function involves the effect of ***coefficient quantization.*** When we set up the system function for a DSP system, say a digital filter, we tacitly assume infinite precision – that is, we don't worry about the precision of the coefficients that are calculated. In fact, there is a limit to the precision, because the digital computer we use to make the calculations has a finite word length, which limits the precision of the calculated numbers. However, the word length is long enough that the limit of the precision doesn't affect the filter response. The calculated filter response exactly satisfies the stated requirements, as far as we can tell.

However, an actual digital filter or other DSP design may not be implemented on a standard digital computer. It may be in a system with an embedded microcontroller or other processor with a word length much smaller than the traditional PC. Therefore, it is instructive to consider the effect of the limited precision caused by the limited word length.

Incidentally, this is exactly parallel to the component tolerance problem in analog circuits. Case history.

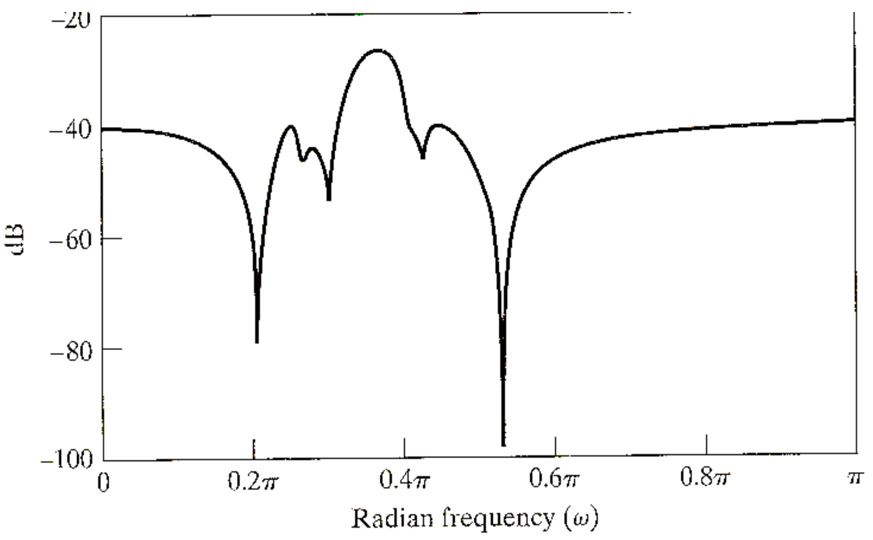
Theoretical and experimental work shows that different configurations respond quite differently to the limited precision resulting from the quantization of the coefficients. In particular, it has been found that the direct forms are quite sensitive to the finite precision of the coefficients. This is caused by the fact that the coefficients affect all the poles and zeros, making the actual performance quite different from the desired result.

Here is the frequency response of a 12th order elliptic filter with coefficients unquantized and a direct form realization:



(a)

With 16 bit coefficients using a parallel form, the response is essentially the same. With a direct form, and 16-bit quantization, the following frequency response results:



It is apparent that the response degrades substantially, and is essentially unusable with a direct form.

For ***FIR*** systems, which depend only on the zeros of the system function, the response is not nearly so degraded by coefficient quantization, and the direct forms are commonly used.

Roundoff noise. Computations made in DSP systems will not, in general give results that can be exactly represented by the finite-length words in the computer being used. The difference between the computed results and the value obtained by rounding off to the nearest valid binary value is treated as noise, and since it varies from one computation to the next, its effect it is usually analyzed as if it is random noise. Statistical analyses are employed to analyze the effect of the noise.

It can be shown that the effect of roundoff noise is influenced by the pairing of the second order sections into which the basic direct form can be converted. Recall that we factored the simple second order system function into two numerator and two denominator factors. Then we separated them into two first order sections by pairing one numerator factor with one denominator factor:

$$H(z) = \frac{1 + 2z^{-1} + z^{-2}}{1 - 0.75z^{-1} + 0.125z^{-2}} = \frac{(1+ z^{-1})(1+z^{-1})}{(1 - 0.5z^{-1})(1-0.25z^{-1})} = \frac{(1 + z^{-1})}{(1-0.5z^{-1})} \frac{(1 + z^{-1})}{(1 - 0.25z^{-1})}$$

We could have adopted a different pairing, which would have not made any difference in this simple case because both numerator factors are the same. However, in the general case it is possible to construct several different pairings, and the effect of roundoff noise is different for each pairing.

In addition, the treatment of noise varies with the configuration of the system function used. "Noise" introduced by the quantization step at each stage of a system is effectively filtered by succeeding stages. Thus the effect of roundoff noise depends on the choice of form – direct, parallel, cascade, etc. Theoretical analysis of this effect is difficult, however, and it is not possible, in general, to predict which form of the system function will give the best results in accommodating roundoff noise.

We are not prepared to go into these factors in further detail,, but will simply state that the treatment of error due to roundoff noise is another reason for considering forms of the system function of a DSP system, other than the basic direct forms.

Overflow. Another reason for considering the ordering of the factors of a system transfer function is the problem of ***overflow.*** As we have noted, two of the basic operations done in DSP are addition and multiplication. In either type of computation, it is entirely possible that the result will be too large for the word size of the computer system involved. Without some corrective action, over flow will cause very serious errors. It can modify the sign bit, or change the magnitude drastically. The order in which the sections of a transfer function are calculated affects the overflow problem, but as with some of the other problems we have noted, it isn't possible, in general, to predict the correct order. Simulation and test cases will probably be necessary.

In general, the usual approach to eliminating the overflow problem is to scale the inputs so that overflow doesn't occur at any node in the system. There are several somewhat formal strategies for choosing where and how to scale the signal levels.

Another approach allows saturation, in which the magnitude of a signal at a node is limited to full scale of the word size, and there is no carry out to the higher bit. The use of floating point arithmetic and a double-length accumulator in the DSP system greatly reduces the overflow problem, as larger operands can be accommodated.

The Fast Fourier Transform. This is another of the major uses of Digital Signal Processing. We used it to compute the frequency content of a filtered and unfiltered signal when we designed filters via Matlab.

As we have seen earlier, the simplified form of the FFT and its inverse are represented by the two equations:

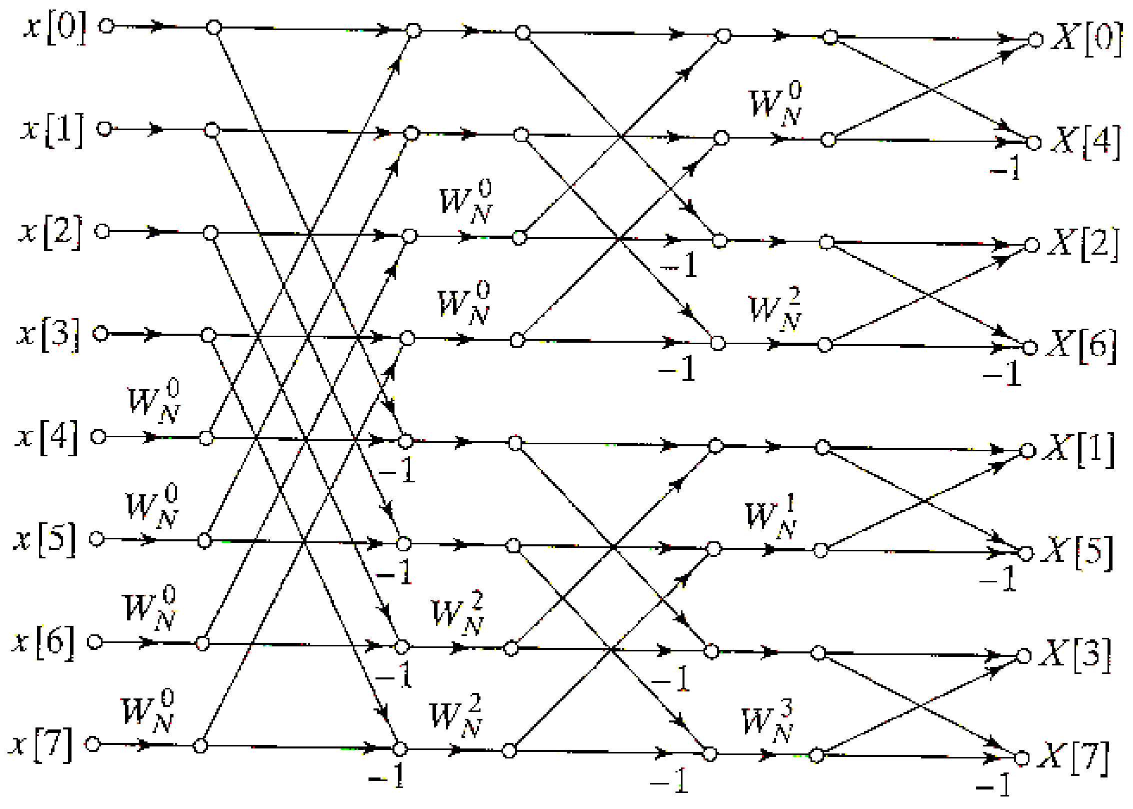
$$x[n] = (1/N) \sum_{n=0}^{N-1} X[k]W_N^{kn}$$ the ***synthesis*** equation

and

$$X[k] = \sum_{n = 0}^{N-1} x[n]W_N^{kn}$$ the ***analysis*** equation

where the notation is simplified by the substitution $W_N = e^{-j(2\pi/N)}$.

In carrying out these two equations, the lengthy computation is broken down into smaller and smaller blocks, ending up with a series of 2 x 2 complex computations. Thus, the format of the computations for the FFT is entirely different from that for a standard linear difference equation or its z-transform. The repeated breaking down of the formulas, called decimation, results in a flow diagram like that below:



The X-shaped operations shown above occur throughout the FFT process and are called ***butterflies.***