Convolution and Correlation

Introduction

The principle behind these two operations is the essence of DSP filter implementations. The basis of convolution and correlation is the ubiquitous 'multiply/accumulate' that all the example systems in these papers have used to date.

Convolution needn't be convoluted!

The starting point is that you have two sampled data signal sequences. For both discrete correlation and convolution, we 'run one set of samples past the other'. Of course we have to pause every sampling period when the sets are lined up and multiply then all out individually. Then we sum them and move on by one sample period and do it all again, repeating until one set is completely past the other. You can see how difficult this is to explain in words although it isn't complicated really, see fig 1. The only difference between correlation and convolution is that for convolution you have to 'flip' over one of the sample sets so that the set is back to front. So why worry about the difference between correlation and convolution? Correlation can be very useful when looking for signals in noise, or when looking for a wanted signal among unwanted ones. However, convolution has a particular advantage in that the time domain act of convolution is the SAME as multiplying in the frequency domain. Because this is going to give us the basis of big computational performance gains we shall concentrate on it.

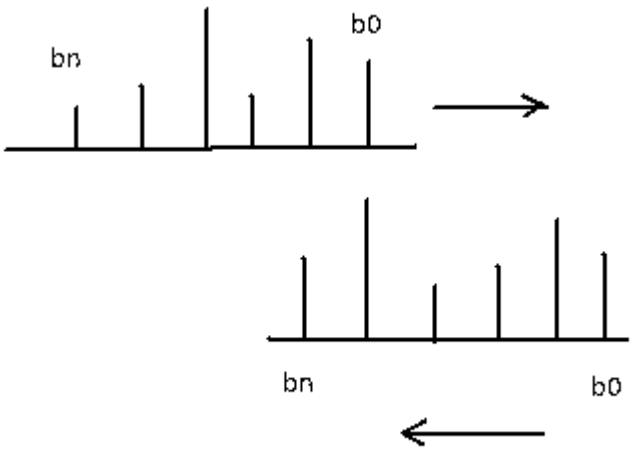

Fig 1 discrete convolution of samples
The usual expression defining this is:

$$y[n]=\sum x[m]*p[n-m]$$

where y(n) is the nth output sample, m is the data set size, x and p are the two sets, *=convolution.

which enables us to see just how many operations are involved. It turns out to be m squared when both data sets are m elements long. For practical reasons the arrays have to be padded with 0's in order to do the multiply/sum without error. In the above each set is 6 elements but when at the beginning, when bn is adjacent to b0, the 5 other elements of each set need to be multiplied by zero. Thus for the above there would be 10 elements in each set (including padding 0's) and the operation would require 100 multiplications, to say nothing of the summations each sample shift. In case you are not sure about the fundamental relevance of convolution, Fig 2 shows the structure required in terms of delays, multiplies and adds. I hope you recognise the architecture that is the foundation of filter systems.

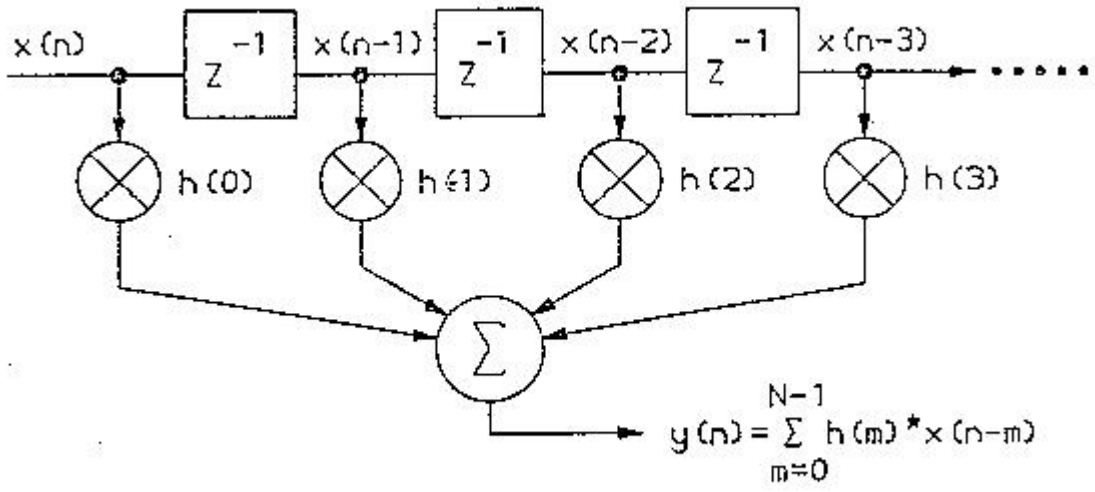

Fig 2 Matched filter (after G.A.King 1990)

The notation is slightly different to that used in previous parts of this series, but z-1 is a single sample delay, X in a circle is multiply and the discrete summation symbol Σ was '+' in previous papers, whilst the two input sample sets are x(n) and h(m).

Fast convolution by Fast Fourier Transform

Remembering that time domain convolution is the same as multiplying the frequency spectra of the two data sets, people wondered if it might be a lot faster to Fourier transform the two sets and then just multiply them together. If there were 10 elements in each set then only 10 multiplies would be needed instead of 100 to do the job in the time domain. This has clear attractions if the job is to be done by software. Formally, fast convolution as it is called uses m log m multiplies (10 for 10 data points in each set). By working in the frequency domain filters could be implemented with just array multiplication. If you are curious about this,it works because whereas time domain samples represent discrete signal values taken at a given moment, when transformed into the frequency domain the frequency components obtained have an existence over all time. Hence no need to worry about the shifting and adding etc. You can then inverse transform the components to get back a time domain output.
Of course this does assume that it is possible to Fourier transform quickly and in fact processors have been built that have Arithmetic units optimised to do this on an array of points in a single

operation. A single dimensional array is called a Vector and Vector Signal Processors have been around for more than fifteen years. In 1990 I used such a processor. It had the ability to implement a 32 stage FIR filter using 128 real samples in one go, and taking just 382 microseconds to do it. That's just under 3 microseconds per sample, or a sampling rate of 333 KHz.

Anyway, Fig 3 below defines the algorithm. As an aside consider the filter in Fig 2. The set h(m) could of course be the filter coefficients derived from the sampled desired impulse response.

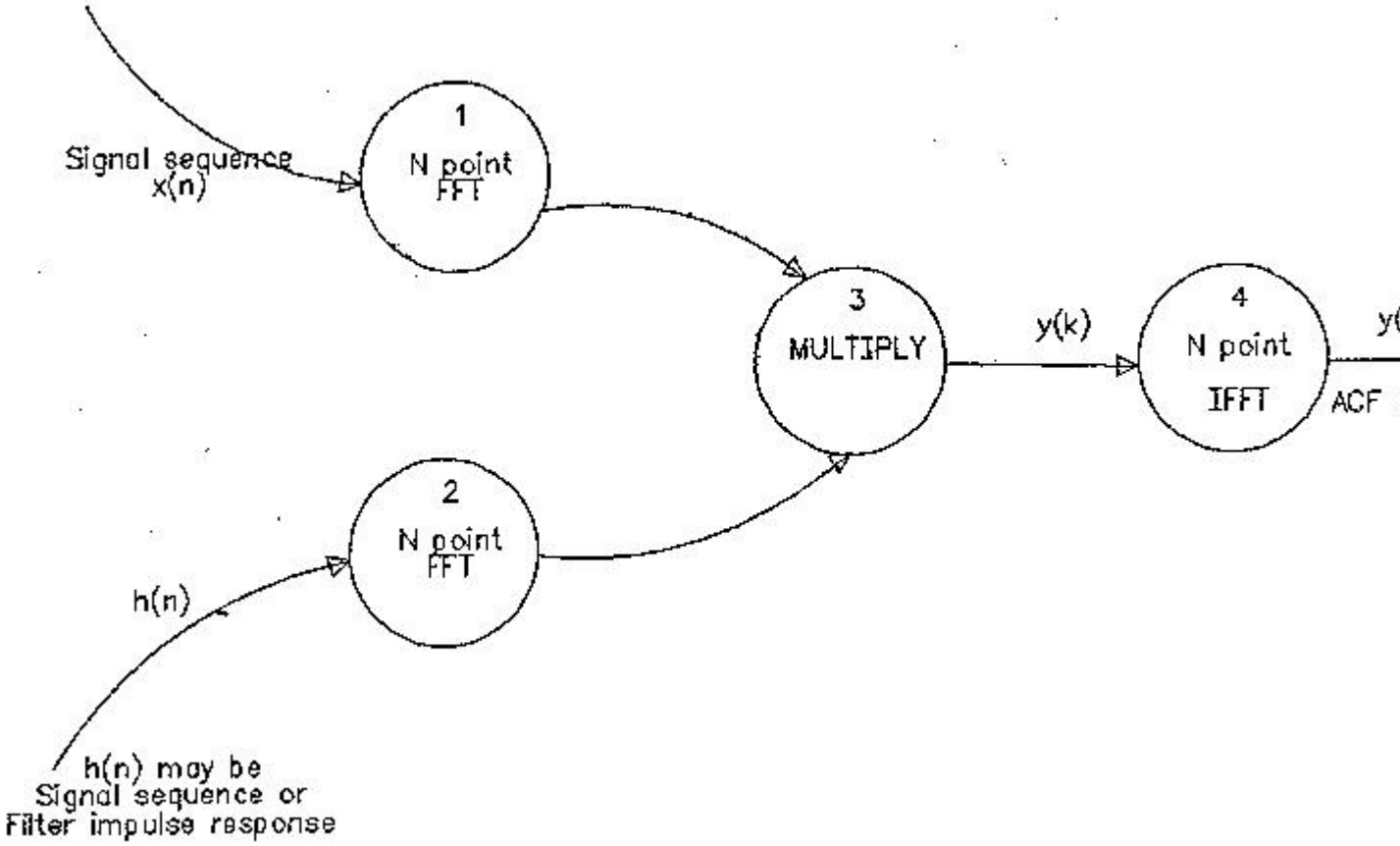

Fig 3 Fast Convolution (G.A.King)

Conclusions

The justification for this leans heavily on the ability to do the algorithm above in substantially less time than the shift,multiply, add option. Again the other presumption is that all the processing is to be done with software running on a dedicated processor. However, it is possible to implement in hardware (to a given level of complexity), using the programmable 'sea of gates' EPLD devices. Remembering the rationale for these articles is to prepare for evaluating PICs and other microcontrollers for use in DSP we shall next look at how to do FFT's.