HIGH SPEED RECURSIVE REALISATIONS OF SMALL LENGTH DFTS

T.E. Curtis
(Admiralty Research Establishment, Portland, Dorset)

and

M.J. Curtis
(University of Essex, Colchester, Essex)

ABSTRACT

This paper develops a prime radix transform algorithm that can be used to calculate the discrete Fourier transform (DFT) of a data block of length P, a prime, using fixed coefficient second order recursive filter sections: this simple hardware structure allows compact, high performance DFT processors to be developed. Further “massaging” of the prime radix algorithm produces the zero factor transform (ZFT) implemented using first order recursive filter sections, with scaling values of -1.

The prime radix and zero factor transforms are limited to data block lengths that are prime: this constraint can be removed by re-factoring the transform equation and implementing the resultant algorithm as a two-stage recursion process. This method, the composite radix Fourier transform (CRAFT), can process data blocks of any length and is particularly suitable for transforming small blocks of data of length equal to 2^n

All of these algorithms can be implemented using first or second order recursive filter structures with fixed coefficient scaling multipliers, and high precision transforms can be produced by exploiting parallel error cancellation techniques. The simple hardware structure that results from this development has allowed high speed transform processors to be built, using both commercially available logic families and gate-arrays of moderate complexity.

1. INTRODUCTION

The discrete Fourier transform (DFT) [1] is one of the fundamental operations in digital signal processing. Many applications require compact, efficient DFT implementations and much has been published in the literature on algorithms to calculate fast Fourier transforms [2, 3], often on general purpose main-frame or minicomputers. However, many real-time applications require either higher throughputs than can be obtained using “conventional” software based algorithms running on minicomputer/array processor combinations or are constrained in terms of the volume and/or power available for the complete system.

In some applications, for example “smart sensor” processing, both of these constraints appear simultaneously. Multi-channel sensor processing applications often require real-time transform systems with optimised hardware-based processing engines to cope with the processing load within the power/size budget of the system. Many of these realisations employ algorithms developed originally for software based systems and gain their speed simply by implementing the “number crunching” processors on “state-of-the-art” hardware. This approach requires that equipment manufacturers have continued access to the enabling high technology components, if they are to continue to improve system performance.

At present, a number of advanced LSI signal processing devices are available from commercial sources in Europe, the USA and Japan for implementing transform systems. However, these devices are often not ideal in specific applications, when it is of often desirable or necessary to integrate the analogue sensor processing on silicon. The resulting high level of integration required for conventional (i.e. complex arithmetic FFT butterfly-based) processors, typically of the order of 80,000 gates, results in large silicon die size and high cost.

The design cycle costs to integrate the predominantly analogue sensor front end with the digital back end processing is also high, and whilst advanced technology and high performance components are being developed in various national and international programmes (e.g. VHSIC in the US, VAD in the UK), they are for predominantly military applications and are unlikely to be made widely available for commercial exploitation.

The cost of developing similar high performance components for the commercial sector is high and consequently systems companies, and other agencies that fund these developments, often prefer to market complete systems rather than individual devices. As a result, the availability of ‘state-of-the-art’ high performance processing elements to OEM users is limited.

The transform techniques outlined in the following sections have been developed to implement low complexity, high performance processors using low-grade, “off-the-shelf” silicon technologies, i.e. in available logic families or in gate array technologies available from commercial foundries in the UK, particularly those with a mixed analogue/digital capability, e.g. BiCMOS. The practical constraints involved in developing this type of hardware-based processor differ significantly from those in the development of software-based systems or those employing custom VLSI designs. In particular, the overall complexity of the implementation must be minimised to reduce silicon area and the number of gates used so that processors can be realised on available gate array based silicon.

The following paragraphs outline the development of two algorithms for the calculation of the DFT of a data block of length equal to a prime or the product of two or more primes, using Goods decomposition [5]. Later sections extend the method to develop a generalised algorithm for transforms with composite block length.

2. THE PRIME RADIX TRANSFORM (PRAT) [4]

The DFT of a block of complex data of length P is given

$$A_r = \sum_{k=0}^{P-1} B_k \exp\{-j\omega kr\} \qquad (1)$$

where $\omega = 2\pi/P$ and $P-1 \geq r \geq 0$.

The data block B_k is usually processed sequentially, with the data used in time order, but this need not be the case. The prime radix transform can be derived by expanding equation (1) and re-factoring. For example, the 7-point transform can be written : -

$$\begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \\ A_6 \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 & W^4 & W^5 & W^6 \\ W^0 & W^2 & W^4 & W^6 & W^1 & W^3 & W^5 \\ W^0 & W^3 & W^6 & W^2 & W^5 & W^1 & W^4 \\ W^0 & W^4 & W^1 & W^5 & W^2 & W^6 & W^3 \\ W^0 & W^5 & W^3 & W^1 & W^6 & W^4 & W^2 \\ W^0 & W^6 & W^5 & W^4 & W^3 & W^2 & W^1 \end{bmatrix} . \begin{bmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \end{bmatrix} \qquad (2)$$

It can be seen from equation (2) that all values of the coefficient vector W^0 through W^6 are used in the calculation of every frequency cell, except the first, i.e. the dc cell A_0. If we treat this as a special term, we can expand equation (2) and re-factorise it so that the coefficients, rather than the data, are used sequentially.

So expanding (2) and re-factorising gives:-

$$\begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \\ A_6 \end{bmatrix} = \begin{bmatrix} B_6 & B_5 & B_4 & B_3 & B_2 & B_1 & B_0 \\ B_3 & B_6 & B_2 & B_5 & B_1 & B_4 & B_0 \\ B_2 & B_4 & B_6 & B_1 & B_3 & B_5 & B_0 \\ B_5 & B_3 & B_1 & B_6 & B_4 & B_2 & B_0 \\ B_4 & B_1 & B_5 & B_2 & B_6 & B_3 & B_0 \\ B_1 & B_2 & B_3 & B_4 & B_5 & B_6 & B_0 \end{bmatrix} . \begin{bmatrix} W^6 \\ W^5 \\ W^4 \\ W^3 \\ W^2 \\ W^1 \\ W^0 \end{bmatrix} \qquad (3)$$

Re-writing this in the summation form of equation (1) gives :-

$$A_0 = \sum_{k=0}^{P-1} B_k \qquad (4a)$$

$$A_r = \sum_{k=0}^{P-1} B_{kr^*} .W^k \qquad (4b)$$

where r* is given by the congruence $r.r^* \equiv 1 \bmod P$.

This has not changed the transform, simply the order in which we carry out the calculation: in the general case, the coefficient vector can increment by a factor Q, with $P\text{-}1 \geq Q \geq 1$, rather than as in equation (3). In this case, the summation form of the algorithm is given by :-

$$A_0 = \sum_{k=0}^{P-1} B_k \qquad (5a)$$

$$A_r = \sum_{k=0}^{P-1} B_{kr^*} .W^{Qk} \qquad (5b)$$

where r* is given by the congruence $r.r^* \equiv Q \bmod P$.

The d.c component of the transform, $A_{0,}$ can be calculated by straightforward accumulation of the data samples: the rth frequency cell can be written in the general form:-

$$A_r = \sum_{k=0}^{P-1} b_k.W^{Qk} \qquad (6)$$

where b_k is the shuffled version of B_{kr*} in equation (4)

Equation (6) can be implemented by expanding it and factorising into partial sums :-

$$A_r = (((\,..\,(((\,b_{P-1}.W^Q + b_{P-2}).W^Q + b_{P-3}).W^Q \ldots$$
$$\ldots + b_2).W^Q + b_1).W^Q + b_0) \qquad (7)$$

when it can be seen that equation (6) represents the first order recursive filter with complex pole W^Q, shown in Figure 1, with the z-domain transfer function:-

$$F(z^{-1}) = 1 / (1 + z^{-1}.W^Q) \qquad (8)$$

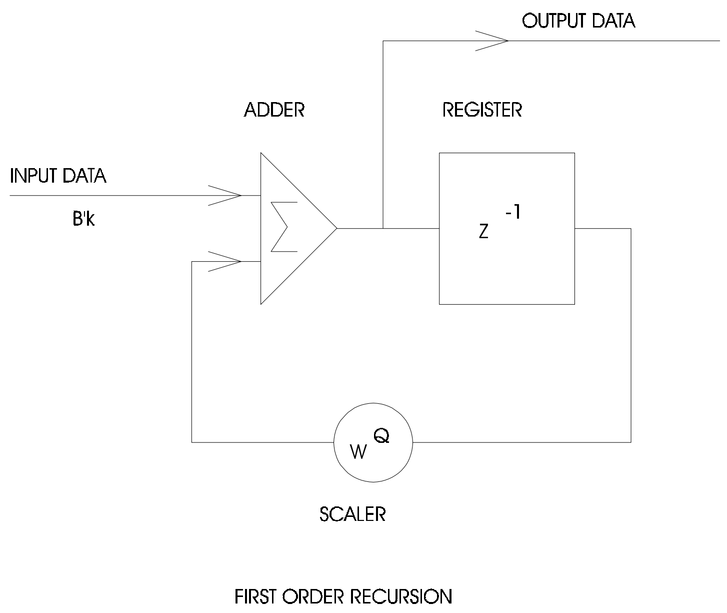

Figure 1- Direct Implementation of Equation 8.
First order recursion with complex pole W^Q

Equation (8) can be implemented directly, but the number of multiplications required in the recursion loop can be reduced by normalising to generate the second order recursive structure : -

$$F(z^{-1}) \;=\; 1 / \;(1 + z^{-1}.W^{Q})$$

$$= \;1 / (1 \;+\; z^{-1}.[\cos\{2\pi Q/P\} - j\sin\{2\pi Q/P\}\;]\;)$$

or :-

$$F(z^{-1}) \;=\; \frac{z^{-1}.(\cos\{2\pi Q/P\} - j\sin\{2\pi Q/P\}\,)}{1 + z^{-1}.2\cos\{2\pi Q/P\} - z^{-2}} \qquad (9)$$

This equation can be realised using the circuit shown in figure 2: the circuit requires only one real multiplier in the recursion loop, multiplication by -1 being trivial in hardware. The recursive operations, corresponding to the demonstrator in equation (9), must be calculated as quickly as possible for high speed operation, as they limit the rate at which data can be clocked into the circuit. However, the output calculations, corresponding to the numerator of equation (9) are only calculated at the transform output rate, and need to be performed much less often (in fact at 1/P times the input rate). Hence a fast combinational multiplier is required in the recursion loop, whilst much slower multipliers, for example a canonic signed serial/parallel implementation, can be used for the numerator.

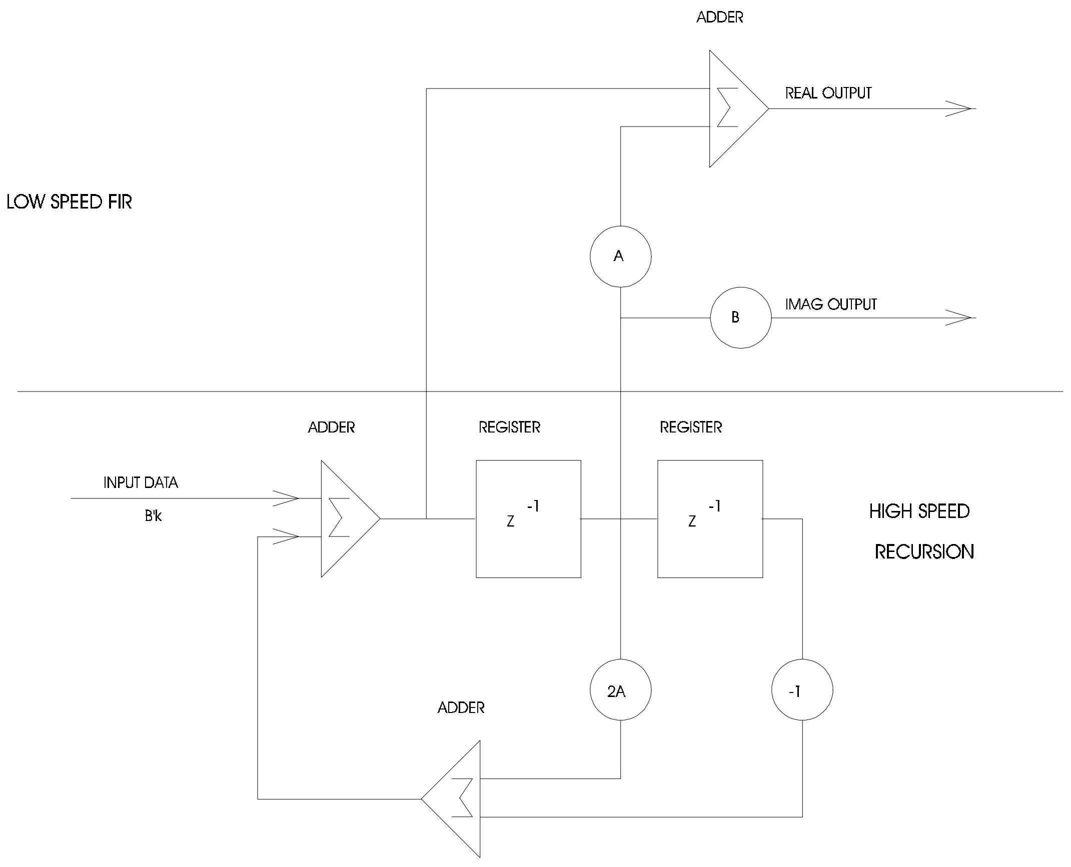

Figure 2 - Direct Implementation of Equation 9.
Second order recursion with real pole

This recursive implementation is similar to that due to Goertzel [6], but the data permutation developed in equation (5) allows us to use a scaling multiplier that is fixed in value throughout the transform calculation, rather than the multi-valued multiplier required by Goertzel. This significantly reduces the hardware complexity as we can, for example, approximate the recursive scaling, d (= 2.cos 2πQ/P), by some close value, D, that can be realised conveniently using binary shift/add network, rather than a multi-valued combinational multiplier.

The error due to the use of an approximate scaling coefficient, D, rather than the exact value, d, can be corrected using a Taylor expansion :-

$$\frac{1}{1 + dz^{-1} - z^{-2}} = \frac{1}{1 + Dz^{-1} - z^{-2}} \cdot F(z)$$

Hence

$$F(z) = \frac{1 + Dz^{-1} - z^{-2}}{1 + dz^{-1} - z^{-2}}$$

Writing D = d + ξ, where ξ is the scaling error gives :-

$$F(z) = \frac{1}{1 - \xi z^{-1} / (1 + Dz^{-1} - z^{-2})}.$$

and :-

$$\frac{1}{1 + dz^{-1} - z^{-2}} = \frac{1}{1 + Dz^{-1} - z^{-2}} \cdot \frac{1}{1 - \xi z^{-1} / (1 + Dz^{-1} - z^{-2})} \qquad (10)$$

Equation (10) can be approximated using the parallel error correction scheme in Figure 3, and high precision prime radix transforms can be produced, with low circuit complexity as in Reference 7.

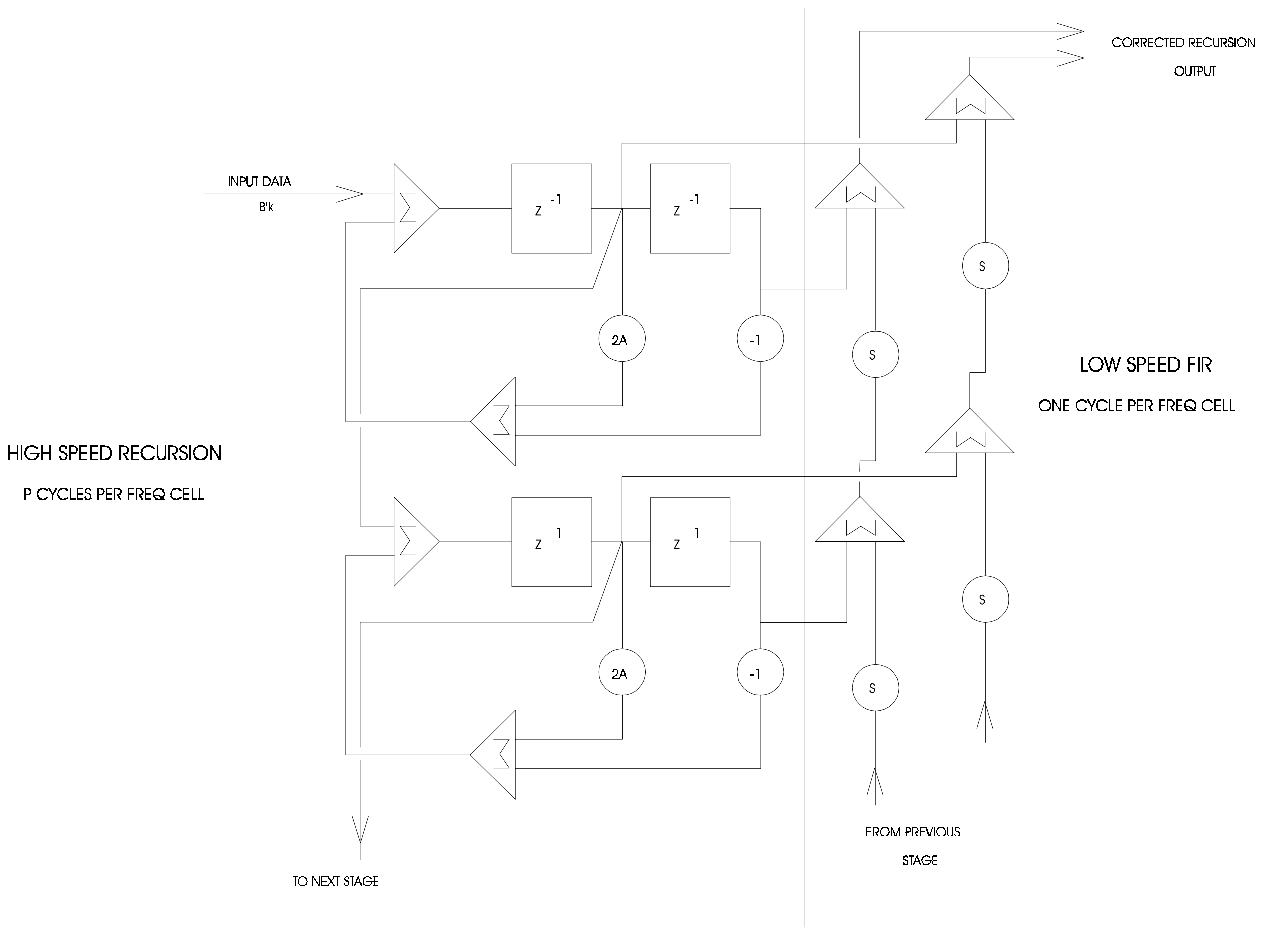

Figure 3 - Parallel Error Correction
Direct implementation of equation 10.

The propagation delays of the scaling circuits in Figure 3 limit the rate at which the PRAT recursion can be clocked in practice, and the following section describes a method to remove the scaling multipliers from the recursion loop and hence improve transform bandwidth.

3. SIMPLIFYING THE PRIME RADIX TRANSFORM

In the previous section, scaling values were considered which could be approximated by simple binary fractions to produce minimum complexity hardware structures.

However, using the parallel error cancellation scheme shown in Figure 3, any scaling value can be approximated to arbitrary precision using the necessary number of parallel error correction stages. Consequently, for minimum complexity recursion hardware, it is convenient to consider transform lengths, P, and phase iteration factors, Q, where the scaling factor can be approximated by zero. This has the advantage that multiplication by zero can be implemented quite quickly and requires no hardware. The second order recursion then reduces to a recursion with scaling factor of -1 and a delay of z^{-2}. Filter decimation techniques can then be applied to the algorithm, allowing further simplification of the recursion structure.

3.1The Zero Factor Transform

The value of d in the prime radix recursion is given by:-

$$d \quad = \quad 2 \,.\, \cos\,(2\pi Q/P) \qquad (11)$$

and $d \Rightarrow$ zero as $2\pi Q/P \Rightarrow \pi/2$.

So, to minimise d, the best values of Q for a given P are given by the integer closest to P/4, i.e. :-

$$Q \quad = \; [\,P \,/\, 4 + 0.5\,]$$

$$\text{or} \quad Q \quad = \; P - [\,P \,/\, 4 + 0.5]$$

where [x] denotes the integer part of X

Hence Figure 3 can be re-drawn with zero multiplies as in Figure 4, with the value of E in the error correction circuit given by :-

$$E \;=\; D \,-\, d$$

$$= \,-\, 2.\cos(\,2\pi\,[\,P \,/\, 4 + 0.5] \;/\, P) \qquad (12)$$

The system in Figure 4 can be implemented directly in hardware using inverting latch/adder loops for the high speed recursion and series/parallel multipliers for error correction, which can be relatively slow, since error correction is only performed at the cell output rate.

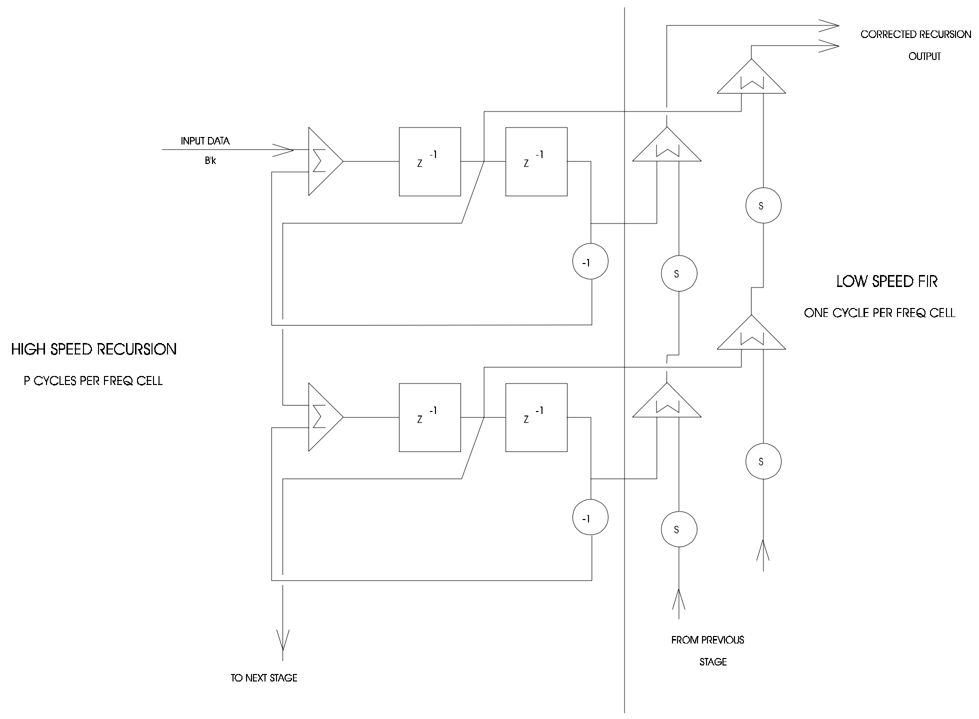

Figure 4- The Zero Factor Transform

3.2 Input Decimation

The recursion system developed in the previous section requires P - 1 recursion cycles. Since the complex plane positions of W^{Q} and W^{-Q} are symmetrical about the real axis, one way to increase transform throughput (and also to reduce form recursion error) is to divide the P - 1 point sequence into forward and reverse rotation recursions, each of (P - 1) / 2 points:-

$$A_{r+} = \sum_{k=0}^{T} b_k . W^{Qk} \tag{13a}$$

$$A_{r-} = \sum_{k=0}^{T} b_{P-k} . W^{-Qk} \tag{13b}$$

Equation (13) can be implemented either using the same recursion hardware to generate two partial sums sequentially or using two separate but identical parallel recursions for higher throughput systems. In either case, the resulting recursion data can be used to generate a pair of frequency cells [4].

As a result of the z^{-2} delay in the PRAT recursion, only alternate input samples are combined in the process. So odd input samples form one component of the recursion output, even input samples the other. Since odd and even samples do not interact, the input sequence can be divided into odd and even parts and processed separately in parallel. This allows a number of other recursion structures to be developed. Figure 5 shows a schematic of a system to operate on decimated input data. As can be seen from the figure, only subtract/latch loops are required to implement the complete recursion. The increased parallelism of this architecture, together with the simpler recursion loop using only first order structures with -1 scaling, allows the transform algorithm to be realised significantly faster than when using the original PRAT system in Figure 3.

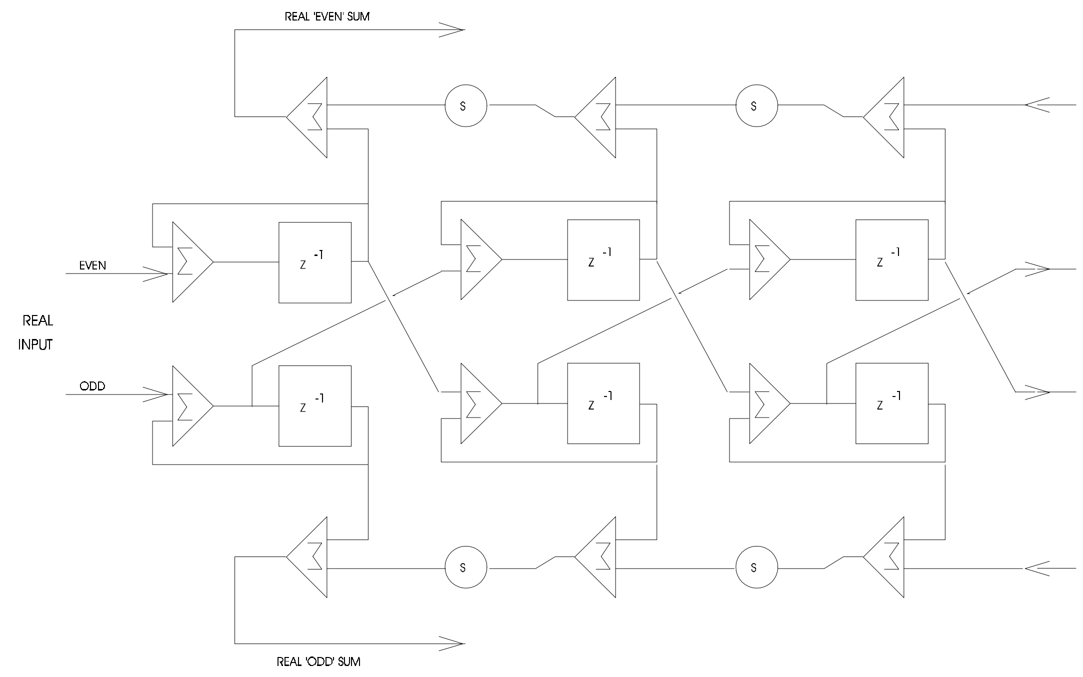

Figure 5 - The Zero Factor Transform using Odd/Even Decimation

Using 74AS or 74F series logic, 29-point ZFTs can be calculated in around 2 useconds and 31-point ZFTs in around 2.5 seconds, allowing 899-point transforms to be calculated in less than 80 microseconds, (using cascaded 29- and 31- point blocks)

4. HIGH SPEED ZFT PROCESSORS

The ZFT processor shown in Figure 6 uses only first order recursions with scaling values of -1. Since this structure is very simple, it is worth considering a FIR implementation of the basic filter structure, rather than the IIR implementation intrinsic to the PRAT algorithm. The recursion can be "unfolded" to give a systolic FIR-like system shown in Figure 6. This uses only adder/subtracters and latches and can potentially provide very high speed transforms.

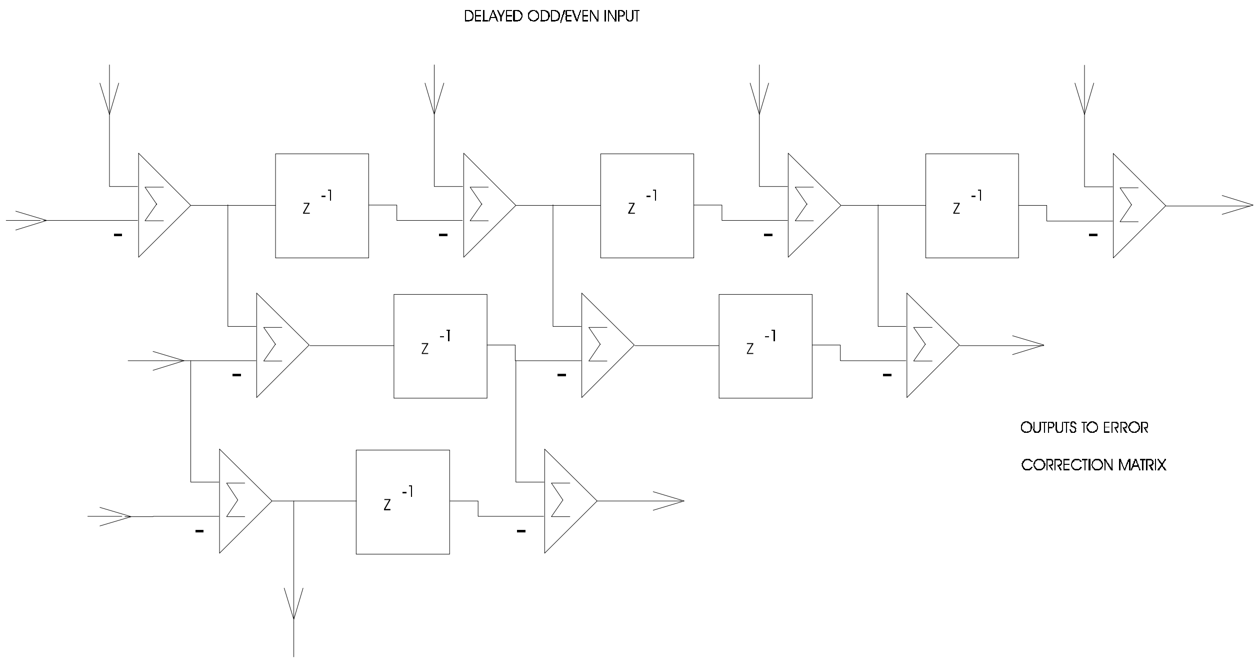

Figure 6- The Peristaltic Implementation of The Zero Factor Transform

The resultant processor, the "peristaltic ZFT" (the ultimate rubbish-in, rubbish-out machine), differs from a systolic processor [8] in that, in the former case, the coefficient data is built into the processor structure, whilst for the systolic processor calculation results from the interaction of separate data and coefficient streams. However, both architectures have a number of common features; both have voracious appetites for data and are difficult to feed efficiently and both get rather messy when the processing pipeline gets blocked. If the peristaltic ZFT can be fed data sufficiently quickly, then potentially it can process data at the rate of two frequency cell outputs per clock cycle.

13-point ZFTs can be calculated in 6 clock cycles, around 120 nseconds using 74AS or 74F series logic: similarly 17-point transforms requires 160 nseconds, so 221-point transforms, using cascaded 13- and 17-point ZFTs, can be calculated in just over 2 useconds, allowing transforms with bandwidths in excess of 100 MHz to be developed. Such transforms have wide application.

The regular structure of the peristaltic ZFT and the simple connectivity between the processing nodes make the system in Figure 6 particularly attractive for VHSIC implementation.

5. THE COMPOSITE RADIX FOURIER TRANSFORM

The transform algorithms discussed so far have been limited to the use of block lengths equal to a prime. For new system designs, this does not create any problems, since many useful transform sizes can be realised using multiple passes through different sized small length transforms, for example, 899-point transforms using cascaded 29- and 31-point PRATs, 221-point using 13- and 17-point PRATs, etc.

However, in some applications, particularly when upgrading existing systems, this constraint to transform size equal to the product of two (or more) moderate sized primes is a problem. The following sections outline a simple modification of the prime radix algorithm to handle other data block lengths. As an example of the approach, we will consider the case when the transform length is equal to 2^n, although the method can be easily extended for any block length.

So for $N=2^n$, say N=3, we can write:-

$$A_r = \sum_{k=0}^{7} B_k .\exp\{ -j\, \omega\, kr\} \qquad (14)$$

where $\omega = 2\pi/8$ and $7 \geq r \geq 0$

Proceeding as previously for the PRAT development, we can write equation (14) in matrix form as:-

$$\begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \\ A_6 \\ A_7 \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 & W^4 & W^5 & W^6 & W^7 \\ W^0 & W^2 & W^4 & W^6 & W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^1 & W^4 & W^7 & W^2 & W^5 \\ W^0 & W^4 & W^0 & W^4 & W^0 & W^4 & W^0 & W^4 \\ W^0 & W^5 & W^2 & W^7 & W^4 & W^1 & W^6 & W^3 \\ W^0 & W^6 & W^4 & W^2 & W^0 & W^6 & W^4 & W^2 \\ W^0 & W^7 & W^6 & W^5 & W^4 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \\ B_7 \end{bmatrix} \qquad (15)$$

It can be seen here that the coefficients W^0 through W^7 are <u>not</u> all used for each frequency cell calculation. In particular, the even cells use sub-sets of coefficients, the odd use them all. In the more general case with transform length N, all coefficients are used when r, the frequency index, is relatively prime to N and subsets are used when it is not.

We can expand equation (15) and again factor it to use the coefficients in any order, say sequentially :-

$$\begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \\ A_6 \\ A_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & (B_0+B_1+B_2+B_3+B_4+B_5+B_6+B_7) \\ B_7 & B_6 & B_5 & B_4 & B_3 & B_2 & B_1 & B_0 \\ 0 & (B_3+B_7) & 0 & (B_2+B_6) & 0 & (B_1+B_5) & 0 & (B_0+B_4) \\ B_5 & B_2 & B_7 & B_4 & B_1 & B_6 & B_3 & B_0 \\ 0 & 0 & 0 & (B_1+B_3+B_5+B_7) & 0 & 0 & 0 & (B_0+B_2+B_4+B_6) \\ B_3 & B_6 & B_1 & B_4 & B_7 & B_2 & B_5 & B_0 \\ 0 & (B_1+B_3) & 0 & (B_2+B_6) & 0 & (B_3+B_7) & 0 & (B_0+B_4) \\ B_1 & B_2 & B_3 & B_4 & B_5 & B_6 & B_7 & B_0 \end{bmatrix} . \begin{bmatrix} W^0 \\ W^1 \\ W^2 \\ W^3 \\ W^4 \\ W^5 \\ W^6 \\ W^7 \end{bmatrix} \quad (16)$$

Equation (16) can be expressed in a general form with the phase rotation given by W^Q, with (N,Q)=l, for any transform size and can be implemented using the two stage recursion system shown in Figure 7. The main second order recursion structure uses either the PRAT or ZFT structures outlined above and a further adder/latch loop accumulates the partial sums of equation (16). Adder/latch accumulation control and data/accumulator selection are provided conveniently by using extra data bits in the PROM address maps. Iterative implementations have been developed in this way for most small length DFTs.

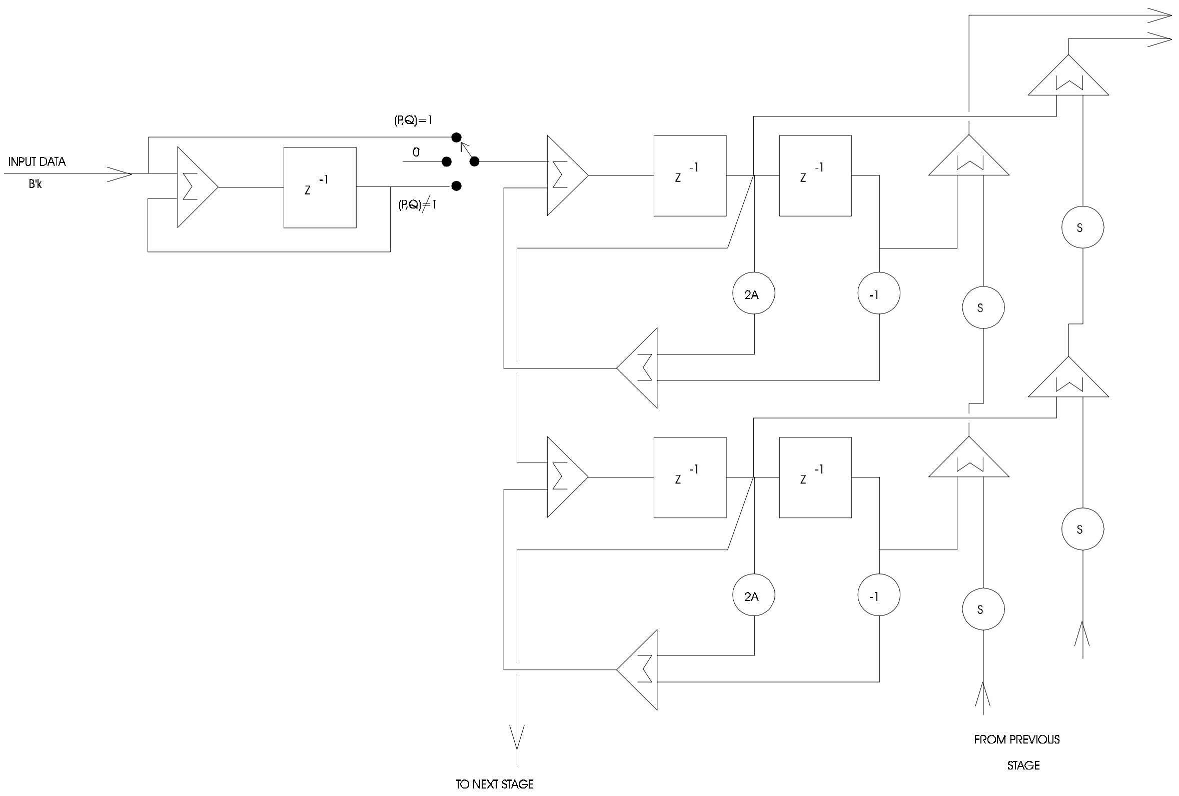

Figure 7- The Composite Radix Fourier Transform (CRAFT)
Direct implementation of equation 16

6. DISCUSSION

A collection of techniques has been outlined which allow Fourier transforms to be implemented efficiently. The simple hardware structures provide transform systems with high throughput and high precision.

Low processor complexity, regular structure and simple inter-processor connectivity allow PRAT, ZFT and CRAFT systems to be developed using available LSI logic families and gate array technologies. The algorithms are sufficiently undemanding in gate count to allow them to be built in MOS, UHS bipolar or GaAs technologies as processor bandwidths and system economics dictate. Moderate throughput systems, using commercial CMOS gate arrays have been developed [7] to provide powerful, high precision DFT processors. These are designed to operate as one of the hardware macros for a high throughput, distributed architecture signal processor to implement high level signal processing graphs for sonar acoustic signal processing [9]. Similar algorithm massaging techniques and simple number theory methods are being employed to develop low complexity processors for other processing nodes in this system, e.g. for filtering, beamforming, matrix manipulation, etc.

7. REFERENCES

[1] Gold, B., and Radar, CM., (1969) Digital Processing of Signals, McGraw-Hill.

[2] Cooley, J.W. and Tukey, J.W., (1965) An Algorithm for the Machine Computation of Complex Fourier Series, Math Comput., 19.

[3] Winograd, S., (1978) On Computing the Discrete Fourier Transform, Math. Comput., 32.

[4] Curtis, T.E. and Wickenden, J.T., (1983) Hardware-based Fourier Transforms: Algorithms and Architectures, IEE Proc, Pt. F, Commun., Radar and Signal Process, 130.

[5] Good, I.J., (1960) The Interaction Algorithm and Practical Fourier Series, J. Roy. Statist. Soc. Ser. B, 1953, 20, Addendum, 22.

[6] Goertzel, G., (1958) An Algorithm for the Evaluation of Finite Trigonometric Series, Am. Math. Monthly, 65.

[7] Spreadbury, D.J. and Rees-Roberts, T.M., (1984) Prime Radix Transforms- From Algorithm to Silicon Implementation, EURASIP Int. Conf. on Digital Signal Processing, Florence, Italy.

[8] Kung, H.T., (1984) Some System and Implementation Issues in Systolic Algorithm Designs, EURASIP Int. Conf. on Digital Signal Processing, Florence, Italy.

[9] Wu, Y.S. et al., (1984) Architectural Approach to Alternate Low-level Primitive Structures (ALPS) for Acoustic Signal Processing, IEE Proc, Pt. F, 131.