MATHEMATICAL APPROACH TO ITERATIVE COMPUTATION NETWORKS

Danny Cohen

USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90291

ABSTRACT

This paper deals with design principles for iterative computation networks. Such computation networks are used for performing repetitive computations which typically are not data-dependent. Most of the signal processing algorithms, like *FFT* and filtering, belong to this class.

The main idea in this paper is the development of mathematical notation for expressing such designs. This notation captures the important features and properties of these computation networks, and can be used both for analyzing and for designing computational networks.

1. INTRODUCTION

The central point of this paper is the application of a precise mathematical notation to express computation networks. This notation captures the concepts of arithmetic operations (such as addition and multiplication) and of timing (e.g., delaying). Once a design is expressed by means of such a mathematical notation, it can be evaluated objectively against a pre-defined set of design objectives, like performance and cost.

The next section, section 2, defines the design objectives which are guiding the examples in this paper. Obviously, other sets of design objectives may be used without deviating from the spirit of the paper.

Section 3 deals with the implementation of an *FIR*-filter, which is a typical signal processing problem. In this section, several designs are suggested and evaluated objectively. In parallel, the mathematical notation to express them is developed.

In this section a design which follows closely the mathematical definition of the *FIR* filter is considered first. Later this design is transformed several times in order to improve it with respect to the predefined design objective.

In this section the graphic representations of these designs are the source of intuition, and their mathematical representations are mainly a means for verifying the correctness of the various transformations of the design.

In section 4 the same technique and the same notation are applied to polynomial multiplication. In this section the mathematical representation is the guiding force, and the graphic representations are used only for demonstration.

In section 5 the same technique is used for polynomial division and for simultaneous multiplication and division of polynomials. In this section the mathematical notation is the only tool used, and the graphic drawings are used as a demonstration only.

It is our conviction that this mathematical notation is a very powerful tool, complementing the intuition which is based on conventional graphic representation.

2. THE DESIGN GOALS

In order to achieve an optimal design, it is necessary to define the design objectives. The following are typically considered to be important:

(*a*) Correctness and accuracy
(*b*) High computation rate
(*c*) Low delay
(*d*) Low parts count
(*e*) Modularity, simplicity, etc.
(*f*) Low power
(*g*) Small size
(*h*) Low cost

Obviously, this is only a partial list. For different applications the relative weights of these objectives may vary. It is generally accepted that (*a*) is the most important, even though we seem to have evidence that this is not always the case.

In some cases (*h*) is the dominant factor, in others it is (*f*) and (*g*). In our discussion, in this note, we consider (*a*) through (*e*), in that priority order.

3. THE FIR-FILTER EXAMPLE

Consider the Finite Impulse Response (*FIR*) filter defined by:

$$y_n = \sum_{i=1}^{N} a_i \, x_{n-i} \qquad (1)$$

This is a non-recursive filter of the *N*th order. Each output (*Y*) is a weighted average of the previous *N* inputs (*X*).

Typically, the *X* sequence is a time series, and the $\{x_i\}$ are available sequentially, starting at x_1, continuing through x_2 and x_3, up to x_M, where typically $M >> N$.

The "edge-effect" at the initialization may be ignored. It is typical to define $x_i \equiv 0$ for $i \leq 0$.

The Z operator

Let Z be the *delay* operator such that:

$$Zx_i = x_{i-1} \qquad (2)$$

In a system which is controlled by a central master clock, this Z operator may be implemented by a simple register.

Similarly, Z^n is defined by:

$$Z^n x_i = x_{i-n} \qquad (3)$$

The Z^n can be implemented by an *n*-stage shift register, which is a *FIFO* (queue).

We will use the following properties of the Z operator:

(*i*) $Z^n F(x,y) = F(Z^n x, Z^n y)$ for $\forall n$, and

(*ii*) if *C* is a constant then $Z^n C = C$ for $\forall n$.

Negative values of *n* mean prediction by $|n|$ steps into the future. Since prediction of external input is not easy to implement, it is advisable to use only $n \geq 0$ when applying the Z^n operator to the input.

The FIR-filter implementation

The expression
$$y_n = \sum_{i=1}^{N} a_i \, x_{n-i} \qquad (1)$$

may also be written as:
$$y_n = \sum_{i=1}^{N} a_i \, Z^i x_n \qquad (4)$$

by using operator-calculus notation, (4) may be written as:

$$Y = \left(\sum_{i=1}^{N} a_i \, Z^i \right) X \qquad (5)$$

For N=4 this means

$$Y = \left(a_1 Z + a_2 Z^2 + a_3 Z^3 + a_4 Z^4\right) X \qquad (6)$$

which can be implemented by the network shown in figure (F1).

The circles in figure (F1) with the a_i's represent the multiplications by the constants which are written inside them.

Checking this network against the design objectives reveals that:

(*a*) Correctness: The correct expression is indeed computed since the values at P_4, P_7, P_9 and P_{10} are Zx_n, Z^2x_n, Z^3x_n and Z^4x_n, respectively.

(*b*) Computation rate: The computation rate is the reciprocal of the computation period, which is the time needed for one multiplication and for adding *N* quantities.

(*c*) Delay: The delay is one Z-period plus the computation period.

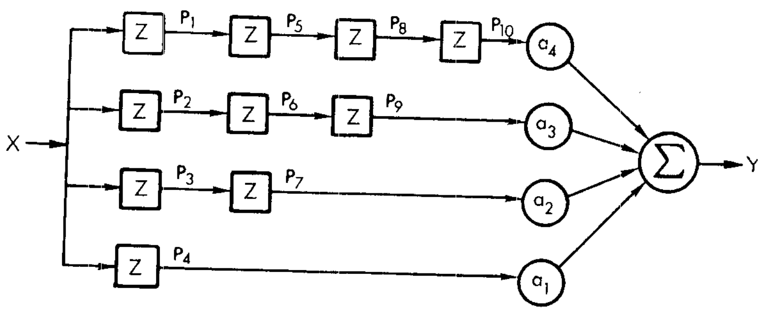

Figure (F1): The implementation of (6).

It is not simple to quantify the parts count, *(d)*, and the modularity objective, *(e)*.

However, the parts count *(d)*, can be improved! Note that the values at P_1, P_2, P_3 and P_4 are all equal to Zx_n. Therefore these points could be unified. Similarly, P_5, P_6 and P_7 could be unified, and so can P_8 and P_9.

This does not change *(a)*, *(b)* and *(c)*, but it does improve *(d)*. The new network is shown in figure (F2).

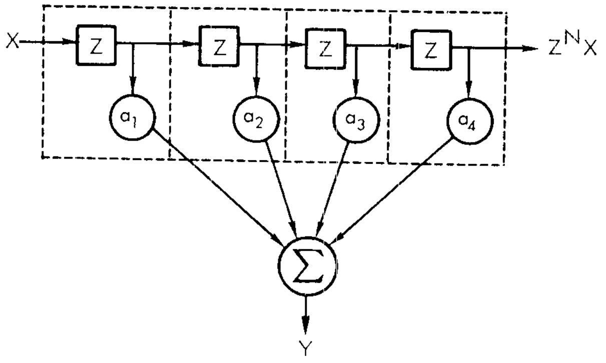

Figure (F2): The improved implementation of (6).

Hence, the parts count, objective *(d)*, is improved by the elimination of 6 delay operators, or $\binom{N}{2}$ in the general case. The modularity, objective *(e)*, is also improved, as seen from the repeated modules, marked by dashed lines in figure (F2).

Improving this design

The *N*-input summation is the Achilles heel of this design, mainly because it does not comply with the modularity requirement.

In addition, the direction of the information flow from the repeating modules into the summation is perpendicular to the direction in which these modules are arranged. This may cause problems with the geometry of the wiring, both in *LSI* and discrete (*IC*'s) implementations, and also on and between printed circuit boards.

In addition, the required number of output lines from any grouping of a set of several modules is proportional to their number, and this may pose severe problems for implementation at any scale.

The way to implement *N*-input summation is by *N*-1 additions. Breaking the summation operation into *N*-1 additions, and dividing them between the modules, as shown in figure (F3), alleviates this problem.

The network shown in figure (F3) is composed of *N* identical modules. This is a great improvement for the design objective *(e)*, modularity.

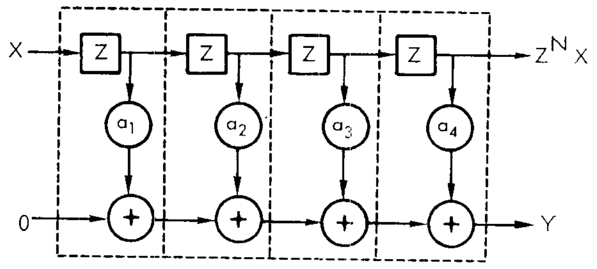

Figure (F3): Using sequential addition.

The left-most adder, the one in the first module (with a_1), does not perform any real addition operation, because one of its inputs always has the zero value. The only purpose of including it in this network is to improve the modularity. Obviously, in discrete implementations, there is no need to include it. Eliminating it does improve the performance and the parts count. On the other hand, in highly integrated implementations, such as *LSI*, having it there is a small price for reducing the number of different modules which are required.

This implementation is represented by

$$Y = \left(\sum_{i=1}^{N} a_i \, Z^i \right) X \qquad (5)$$

In order to improve the delay involved in this computation notice that

$$Z^{-1} Y = \left(\sum_{i=1}^{N} a_i \, Z^{i-1} \right) X = \left(\sum_{i=0}^{N-1} a_{i+1} \, Z^i \right) X \qquad (7)$$

Only non-negative powers of Z are used for the input values (*X*). The "prediction" (Z^{-1}) is applied only to the output (*Y*). It means that at the *n*th cycle (i.e., when x_n is given) the next *Y* value, y_{n+1}, is available.

This is easy to observe from rewriting (6) as:

$$y_{n+1} = a_1 \, x_n + a_2 \, x_{n-1} + a_3 \, x_{n-2} + a_4 \, x_{n-3} \qquad (8)$$

and rewriting (7) as:

$$y_n = a_1 \, x_{n-1} + a_2 \, x_{n-2} + a_3 \, x_{n-3} + a_4 \, x_{n-4} \qquad (9)$$

Both (7) and (9) yield the implementation shown in figure (F4).

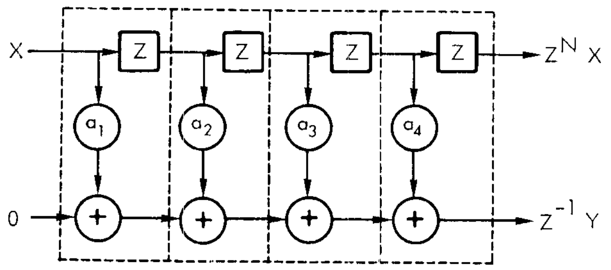

Figure (F4): The implementation of (7).

Note that in figure (F4) the left-most adder (in the first module) is redundant, as mentioned before. So is the right-most delay (in the last module) which does not tax the performance. It also may be eliminated in discrete implementations, but in integrated implementations it is not advisable to do so.

About notation

First, let us introduce another notation, $\prod(X,Y)$, representing the multiplication of X and Y. The purpose of this notation, compared with the usual XY notation, is to make the multiplication operation explicit in the notation, and to distinguish between it and the application of operators.

Note the difference between the following expressions:

$$y_n = \sum_{i=1}^{N} \prod \left(a_i, x_{n-i}\right) \qquad (1)$$

$$Y = \left[\sum_{i=1}^{N} \prod \left(a_i, Z^i\right) \right] X \qquad (5)$$

and the following expressions:

$$y_{n+1} = \sum_{i=0}^{N-1} \prod \left(a_{i+1}, x_{n-i}\right) \qquad (7)$$

$$Z^{-1}Y = \left[\sum_{i=0}^{N-1} \prod \left(a_{i+1}, Z^i\right) \right] X \qquad (10)$$

The first ones, which require unnecessary delay, have the summation range of $[1,N]$, which is the "standard" way for mathematicians for expressing a set of N objects, whereas the last two use the range $[0,N-1]$, which seems to be less "convenient", but yields better delay characteristics.

This illustrates the need to beware of "mental-traps" which may be caused by notation.

Improving the operation rate

The major deficiency of all the networks considered so far is their operation rate, objective *(b)*. As noted before, the operation period cannot be shorter than the time required for multiplication and addition of N quantities.

Even when the multipliers are arranged such that the multiplication time overlaps the addition time, there is still the requirement for the addition to propagate through N (or N-1) stages.

Since N may be very large, it is desirable to eliminate the need for this long addition. This is easy to achieve, by following the "carry-save" idea which uses extra delays in order to improve the data rate. In our problem we introduce delay units between the modules, which delays the output by N cycles but improve the rate.

The resulting network is shown in figure (F5).

Note that the network in figure (F5) is implemented by using the very same modules as in figure (F4) and additional delays. Each of these modules, with the additional delays, is shown in figure (F6).

Since three delays were added (for $N=4$), the result which was $Z^{-1}Y$, in figure (F4), is delayed by Z^3 and is now $Z^3(Z^{-1}Y) = Z^2Y$, and Z^{N-2} in general.

The rigorous proof that the output is correct, is its computation. Let S_j denote the output of such a network, as (F5), with j modules. The output of (F5) is therefore $S = S_4$. We will prove that in general the output of an N-modules network is

$$S = S_N = \left[\sum_{i=1}^{N} Z^{N-i} \prod \left(a_i, Z^{2i-2}\right) \right] X \qquad (11)$$

From the structure of the network and the modules, as shown in figure (F6), we get the following relation

$$S_j = Z\, S_{j-1} + \prod \left(a_j, Z^{2j-2}\right) X \qquad (12)$$

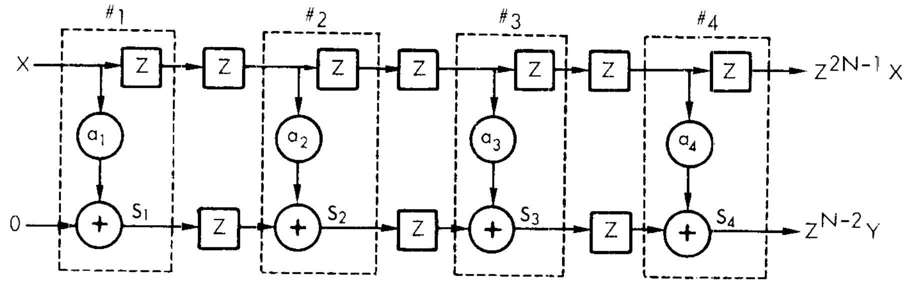

Figure (F5): Implementing the "carry-save" idea.

(11) is proved by induction, starting from $S_0=0$. Assume that it holds for S_{N-1}, and use (12) to evaluate S_N:

$$S_N = Z\, S_{N-1} + \prod\left(a_N,\ Z^{2N-2}\right) X =$$

$$= Z\left[\sum_{i=1}^{N-1} Z^{N-1-i} \prod\left(a_i,\ Z^{2i-2}\right)\right] X +$$

$$+ \prod\left(a_N,\ Z^{2N-2}\right) X =$$

$$= \left[\sum_{i=1}^{N-1} Z^{N-i} \prod\left(a_i,\ Z^{2i-2}\right) + \prod\left(a_N,\ Z^{2N-2}\right)\right] X =$$

$$= \left[\sum_{i=1}^{N} Z^{N-i} \prod\left(a_i,\ Z^{2i-2}\right)\right] X \qquad \text{Q.E.D.} \qquad (13)$$

If the proof seems too rigorous, one can obtain (11) directly by numbering the modules from left to right. In the *i*th module, a_i is used, multiplied by $Z^{2i-2}X$ (X at module 1, Z^2X in module 2, Z^4X in module 3 and Z^6X in module 4), the product is then delayed by Z^{N-i} (here, Z^3 for module 1, Z^2 for module 2, etc.). Hence, the output, S, is the sum of these products $a_iZ^{2i-2}X$, each delayed by Z^{N-i}, as indicated by (11).

Direct methods, compared with rigorous proofs, are simpler and more intuitive, but require caution. Intuition is known to have been misleading, on occasions.

(11) can be simplified to yield:

$$S = \left[\sum_{i=1}^{N} Z^{N-i} \prod\left(a_i,\ Z^{2i-2}\right)\right] X =$$

$$= \left[\sum_{i=1}^{N} \prod\left(a_i,\ Z^{N+i-2}\right)\right] X =$$

$$= \left[Z^{N-2} \sum_{i=1}^{N} \prod\left(a_i,\ Z^{i}\right)\right] X = Z^{N-2}\, Y \qquad (14)$$

Check this network against the design objectives:

(*a*) Correctness: The correct expression is computed indeed, as shown by (14).

(*b*) Computation rate: The computation period is now the time required for a single multiplication followed by an addition, independent of the magnitude of N. Since it is easy to overlap the execution of the multiplication and the addition we do not attempt to separate them even though this may improve slightly the computation period and the computation rate.

(*c*) Delay: The computation delay is equal to (N-2) computation cycles, as is shown by (14).

(*d*) Parts Count: The same number of adders and multipliers, as before, is needed. However, 3 delays are needed in each module. Hence, the total parts count is higher (i.e, worse) than before.

(*e*) Modularity: The modularity is not as good as it used to be in the network shown in figure (F4), which includes only components which are included in the repeated modules.

In order to improve the modularity, we merge the new delays into the old modules. In order not to introduce additional delays, we include in each module the delay which is on its right on the "upper" line, and the delay which is on its left on the "lower" line. Hence, the network implementation now is composed of N modules, each as shown in figure (F6), without the need for any additional components.

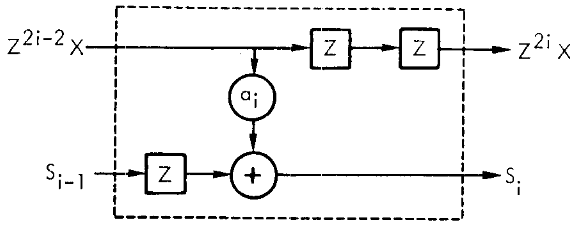

Figure (F6): The *i*th module.

By using a network which consists of N modules as shown in figure (F6), the rate is the best which can be achieved (without separating the multiplication from the addition) and the delay is proportional to N.

Another look

At this time we would like to ask if the reader has noticed that a very important design decision was made without any justification or even discussion. Please take a moment and recall what has been done so far, and look for that important design choice which was made as if no alternative existed.

This design decision is the sequentialization of the summation-operator. We introduced it as a left-to-right sequence of adders without considering other possibilities.

We can use a tree-structure, with $\log_2 N$ depth. Here the carry chain is only $\log_2 N$ long, which is better than N, but still might be too long. The same "carry-save" approach may be used again, by using the delay operation, Z, between every pair of successive adders.

How does this design check against the objectives?

(*a*) Correctness: The correct expression is computed indeed, as shown before.

(*b*) Computation rate: The rate is optimal. As before we do not wish to split the addition from the multiplications.

(*c*) Delay: The delay is only $\log_2 N$.

(*d*) Parts count: The total number of adders required for adding N numbers is N-1, whether they are arranged in linear order or in a tree structure. Hence no change in the number of adders is needed. However, the number of the required delay elements is improved.

(*e*) Modularity: The adders' binary tree is again perpendicular to the data flow and imposes a severe geometrical problem.

By using the modules shown in figure (F7), one can build this network, by having N type-***A*** modules arranged in a linear order, and (N-1) type-***B*** modules arranged in a binary tree structure.

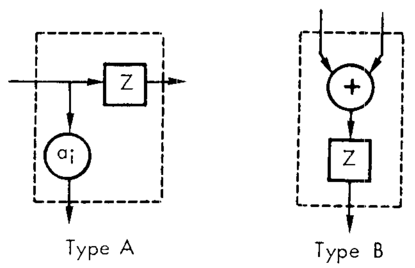

Figure (F7): Modules for the tree implementation

And another look

We have considered the left-to-right and the binary tree arrangements. Let us consider next the right-to-left option. At first, it does not appear to be different from the left-to-right, but it is worth verifying.

Let's look at the network shown in figure (F4), with the direction of the addition reserved. The resulting network is shown in figure (F8).

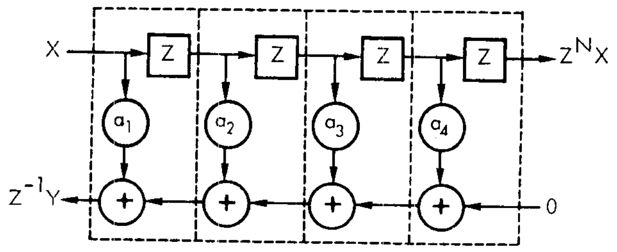

Figure (F8): The right-to-left addition.

Note that the networks shown in figures (F4) and (F8) are identical, and therefore the latter suffers from the same problem that the former does.

The very same "carry-save" idea can be used again, by adding delays. This results in the network shown in figure (F9).

This new network, of figure (F9) has also to be checked against the design objectives.

Starting from (*a*), the correctness, we compute the value of the output S, by using the same technique of numbering the modules from left to right. Now we get:

$$S = \left[\sum_{i=1}^{N} Z^{i-1} \prod \left(a_i,\ Z^{2i-2}\right) \right] X \qquad (15)$$

Note that this is very similar to (11), except that the output of the ith module is delayed now by Z^{i-1} instead of by Z^{N-i} as before, when it was added to the right.

The simplification of (15) yields:

$$S = \left[\sum_{i=1}^{N} Z^{i-1} \prod \left(a_i,\ Z^{2i-2}\right) \right] X =$$

$$= \left[\sum_{i=1}^{N} \prod \left(a_i,\ Z^{3i-3}\right) \right] X =$$

$$= \left[Z^{-3} \sum_{i=1}^{N} \prod \left(a_i,\ Z^{3i}\right) \right] X \qquad (16)$$

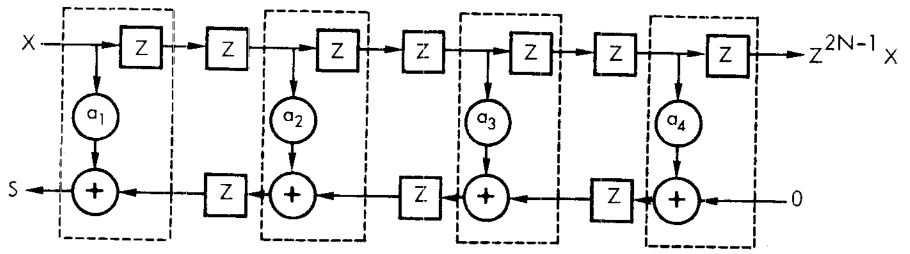

Figure (F9): Right-to-left addition with delays.

Which obviously is not the desired Y. Therefore, the network shown in figure (F9), does not perform the correct computation.

What is the reason that the very same approach which worked so well in the network shown in figure (F5), fails now?

The reason is very simple indeed. In both cases the delays between the adders (on the "lower" line) are needed in order to make the computation period independent of *N*. The purpose of the other delays (on the "upper" line) is to compensate for the delays on the "lower" line, such that the addition is performed coherently.

Since in the left-to-right network (F5) data flows on both lines (the "lower" and the "upper") in the same direction, the same delays have to be introduced in both, to keep the data "in-step".

However, in the right-to-left network (F9) data flows on these lines in opposite directions. Hence, in order to compensate for a delay on the "lower" line, data should be accelerated on the "upper" line. Since Z is used on the "lower", Z^{-1} should be used on the "upper".

It is unfortunate that the Z^{-1} operation is a prediction which we cannot implement in the general case. However, in this case each Z^{-1} happens to follow a Z, such that each cancels the effect of the other.

Let us replace on the "upper" line all the inter-module Z operators by Z^{-1}. This cancels the effect of the intra-module Z operators, such that no delays are needed on this line.

Figure (F10) shows the modified network.

Again, the new design has to be checked against all the design objectives.

Starting with (*a*), the correctness, we get:

$$S = \sum_{i=1}^{N} Z^{i-1} \prod \left(a_i, X\right) = Z^{-1} \sum_{i=1}^{N} Z^{i} \prod \left(a_i, X\right) =$$

$$= \left[Z^{-1} \sum_{i=1}^{N} \prod \left(a_i, Z^i\right) \right] X = Z^{-1} Y \qquad (17)$$

This proves the correctness and also shows that there is no delay whatsoever. We also know that the computation period is minimal since it is equal to the longest "atomic" operation. The parts count is lower than in any other design, and the network is modular.

Based on the above, this design is optimal with respect to correctness, (*a*), computation rate, (*b*), and delay, (*c*), and it also scores highly in the parts count, (*d*), and modularity, (*e*), categories.

An alternative way to draw this network is shown in figure (F11). Note that the addition is performed, again, in the left-to-right direction, because the order of the a_i's is reversed.

Applying the Z-notation to design evaluation

We will show that the Z-notation can be used for the evaluation of all the networks shown before, from figure (F1), to figure (F10). We also claim that this transformation can (and should) be performed without the aid of figures and intuition.

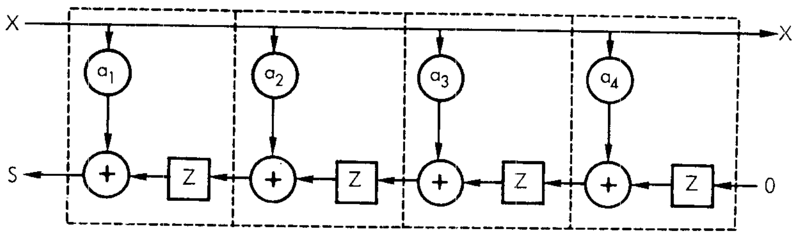

Figure (F10): The modified right-to-left network.

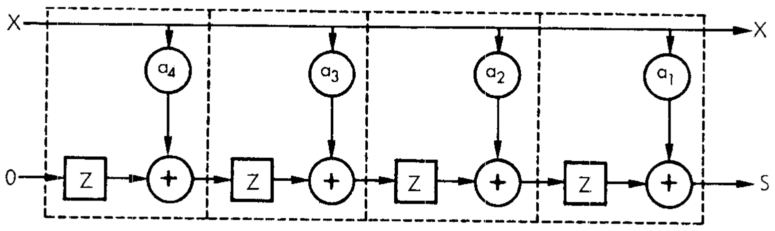

Figure (F11): An alternative drawing of figure (F10)

Let us review the systems which we had so far.

System (*A*) is the one which resulted directly from the definition, and is shown in figure (F1), through figure (F3). Its representation is

$$\text{System (A):} \quad Y = \left[\sum_{i=1}^{N} \prod \left(a_i, Z^i \right) \right] X \qquad (18)$$

Using our experience with this kind of network, we noted that one delay could be saved and we transformed this network into system (*B*) which is the one shown in figure (F4). Its representation is:

$$\text{System (B):} \quad Y = \left[Z \sum_{i=0}^{N-1} \prod \left(a_{i+1}, Z^i \right) \right] X \qquad (19)$$

Then, in order to improve the rate, we further transformed the network into system (*C*), the one shown in figure (F5), whose representation is:

System (C):

$$Y = \left[Z^{-(N-2)} \sum_{i=1}^{N} Z^{N-i} \prod \left(a_i, Z^{2i-2} \right) \right] X \qquad (20)$$

Then we introduced the right-to-left addition, and were able to transform this system into system (*D*), the one shown in figure (F10), whose representation is:

$$\text{System (D):} \quad Y = Z \sum_{i=1}^{N} Z^{i-1} \prod \left(a_i, X \right) \qquad (21)$$

Next, we compare and evaluate these systems, by using their representations, without referring to the figures.

(*a*) Correctness: From the representation above it is evident that all of these systems perform the correct computation.

(*b*) Rate: Both (*A*) and (*B*) require adding *N* quantities at once. Therefore, their computation period is equal to the time required for a multiplication followed by the addition of *N* numbers, where (*C*) and (*D*) require only the time needed for a multiplication and a single addition.

(*c*) Delay: in (*A*) y_n is available in the same cycle as x_n. We use this for delay reference, and denote it as zero delay.

In (*B*) the entire expression, on the right hand side, is multiplied by Z. This means that the output of the network which computes this expression has to be delayed one cycle, in order to have the same delay as in (*a*), the zero delay. Hence, without this additional delay, the output, *Y*, is advanced by one cycle, and is equal to minus one cycle. This means it is earlier by one cycle than (*A*).

On the other hand, (*C*) requires $Z^{-(N-2)}$ in order to achieve the same delay. Since this is not feasible to implement, the Y computed by this network is delayed by (*N*-2) cycles, compared with (*A*).

(*D*) has, obviously, the same delay as (*B*). Thus, (*D*) also is earlier by one cycle than (*A*).

In summary, in the general case, the delays are:

System implementation	*A*	*B*	*C*	*D*
Delay (in cycles)	0	-1	*N*-2	-1

However, even though both (*B*) and (*D*) have the same delay in cycles, (*D*) has a smaller delay since its cycle is shorter. Hence, in this implementation, y_{n+1} is available a shorter time after x_n is given, compared with (*B*).

(*d*) Parts count: The modular implementations, including the additional delays and the additional adders (which may be required on either end of the network in order to achieve the modularity), are compared with each other.

All four implementations require *N* multipliers, and *N* adders (or *N* multiply-&-add units). They differ only in the delay requirements.

Both (*A*) and (*B*) require *N* delays for *X*.

(*C*) requires 2*N* delays for *X*, and *N* delays for the partial sum of the products. These delay units require, in general, more capacity (bits) than for delaying *X*, especially if fixed point arithmetics is used.

(*D*) requires N delays, also for the partial sum of the products.

(*e*) Modularity and simplicity: All four implementations are equally modular, with the same level of complexity.

The rating of these systems is summarized in the following table. *S* > *T* means that *S* is better than *T*.

(*a*) Correctness	(*A*) = (*B*) = (*C*) = (*D*)
(*b*) Data rate	(*C*) = (*D*) > (*A*) = (*B*)
(*c*) Delay	(*D*) > (*B*) > (*A*) = (*C*)
(*d*) Part count	(*A*) = (*B*) > (*D*) > (*C*)
(*e*) Modularity	(*A*) = (*B*) = (*C*) = (*D*)

This shows that (*D*) is the best, if performance is the major objective, but (*B*) is the best if the parts count is the major one.

4. APPLICATIONS FOR POLYNOMIALS MULTIPLICATION

The previous example, the *FIR* filter, was designed by using intuition to operate on computation networks represented by drawings. The Z-notation could be used, but is less intuitive.

Next we compute multiplication and division of polynomials, and design computation networks to implement these operations. However, now we use the Z-notation for the design of the networks, and use diagrams only to demonstrate the design.

The problem of polynomials multiplication

Let $A(t)$ and $X(t)$ be polynomials in t, of degrees c and m, respectively

$$A(t) = \sum_{i=1}^{c} a_i\ t^i \quad ; \qquad X(t) = \sum_{i=0}^{m} x_i\ t^i \tag{22}$$

Let $Y(t)$ be the product polynomial of $A(t)$ and $X(t)$.

$$Y(t) = \sum_{i=0}^{m+c} y_i\ t^i = \left(\sum_{i=0}^{c} a_i\ t^i\right)\left(\sum_{i=0}^{m} x_i\ t^i\right) \tag{23}$$

By equating the coefficients of t^i we get

$$y_n = \sum_{i=0}^{c} a_i\ x_{n-i} \ \left(x_i \equiv 0 \text{ for } i < 0 \text{ and } i > m\right) \tag{24}$$

We are interested in finding the coefficient set of the polynomial $Y(t)$, from the given coefficient sets of $A(t)$ and $X(t)$. We are not interested in evaluating any of these polynomials for particular values of t.

In many applications $A(t)$ is a fixed polynomial, and $X(t)$ is a variable one. The computation problem is to compute the $m+c$ coefficients of $Y(t)$, from the given m coefficients of $X(t)$, and the fixed c coefficients of $A(t)$.

Since (24) is identical to (1), except for the boundary condition and the range, the same networks which compute the *FIR* filter, can also perform this polynomial multiplication.

Since (24) contains a_0, one more stage is needed, and the computation is performed such that y_n is available in the cycle when x_n is given. In other words, the delay now is 0, instead of the -1 cycle as we had before.

Figure (F12) shows the network for this computation. Note that it starts with a_0 (compared with a_1 in the previous network) and that its output is Y (compared with $Z^{-1}Y$ before). Because of the boundary conditions it is important to clear all the delay units before starting the operation, and to provide $x_i = 0$ for $i = m+1, m+2, \ldots, m+c$. When these values are given, the last c values of Y are obtained. Since there are $m+c$ values of Y, and only m values of X, this "runout" operation is indeed expected.

The initial clearing can be performed, just like the runout operation, by proving the network with c zero-values for X. During this period the obtained Y values are invalid.

Obviously, this network is represented by:

$$Y = \sum_{i=0}^{c} Z^i \prod\left(a_i, X\right) \tag{25}$$

Reversing the order of X

In several applications it is preferred that x_n is available before x_{n-1}. In these cases x_m is leading and x_0 trailing.

If this order is used then the operator Z has a predicting role, and Z^{-1} is a delay. Since (25) is implemented with positive powers of Z, another implementation which uses only negative powers of Z is needed.

Multiply (25) by Z^{-c} and get

$$Z^{-c}\ Y = \sum_{i=0}^{c} Z^{i-c} \prod\left(a_i, X\right) = \sum_{i=0}^{c} Z^{-(c-i)} \prod\left(a_i, X\right) =$$

$$= \sum_{j=0}^{c} Z^{-j} \prod\left(a_{c-j}, X\right) \tag{26}$$

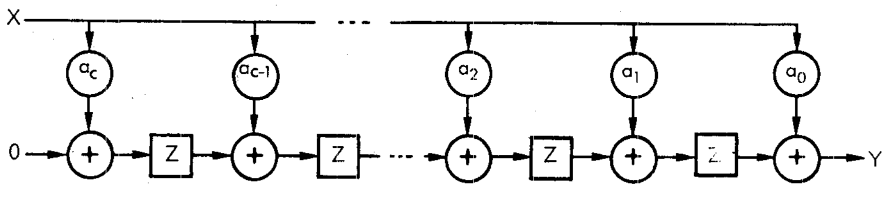

Figure (F12): Polynomials multiplication.

Since this has the same structure as (25) the same network can be used to perform this operation, except that (*i*) Z^{-1} is used instead of Z. However, since Z meant a delay before, and Z^{-1} means a delay now, this is no real change of function, only of labeling, (*ii*) the order of the a_i's is reversed, because we have now a_{c-j} where we had a_i before, and (*iii*) the output now is $Z^{-c}Y$ instead of Y, as before. This means that when x_n is given to the network, y_{n+c} is available. Therefore, when x_m, the leading coefficient of X, is made available to the network, then y_{m+c}, the leading coefficient of Y, is computed. The resulting network is shown in figure (F13).

Computing sum of polynomials products

Consider the problem of computing $W(t)$, which is defined by:

$$W(t) = A(t)\,X(t) + B(t)\,Y(t) \qquad (27)$$

Where $A(t)$ and $B(t)$ are of degree c, and $X(t)$ and $Y(t)$ are of degree m. Obviously, $W(t)$ is of degree $m+c$.

By using (26) we may get:

$$Z^{-c}\,W = \sum_{j=0}^{c} Z^{-j}\prod\left(a_{c-j},X\right) + \sum_{j=0}^{c} Z^{-j}\prod\left(b_{c-j},j\right) \qquad (28)$$

This yields, for c=3, the network shown in figure (F14). However, (28) may be written also as:

$$Z^{-c}\,W = \sum_{j=0}^{c} Z^{-j}\left[\prod\left(a_{c-j},X\right) + \prod\left(b_{c-j},Y\right)\right] \qquad (29)$$

which yields the combined network shown in figure (F15).

5. DIVISION OF POLYNOMIALS

Polynomial division is obviously the inverse of the polynomial multiplication. The division is defined in the usual way, by the relation:

$$Y(t) = A(t)\,X(t) \qquad (a_c \neq 0) \qquad (30)$$

where $A(t)$ and $Y(t)$, are given polynomials of degree c and $m+c$ respectively. $X(t)$ which is to be determined is a polynomial of degree m.

Division, unlike multiplication can be performed only by starting with the most significant (highest power) of Y. This non-symmetry is due to requiring only that the leading coefficient of $A(t)$ must not be zero.

Therefore, we use (26) and not (25) in order to invert the multiplication.

Equation (26) states:

$$Z^{-c}\,Y = \sum_{i=0}^{c} Z^{-i}\prod\left(a_{c-i},\,X\right) \qquad (26)$$

Since the operation has to be performed from the most significant to the least significant term, at any stage in the computation of $X(t)$, the higher order terms of $X(t)$ must already be known.

Therefore, we seek to express X by using A, Y and $Z^{-i}X$ for positive values of i, but not including $i=0$.

Extract Z^0X from (26) and get:

$$Z^{-c}\,Y = \prod\left(a_c,\,X\right) + \sum_{i=1}^{c} Z^{-i}\prod\left(a_{c-i},\,X\right) \qquad (31)$$

Isolate it and get:

$$\prod\left(a_c,\,X\right) = Z^{-c}\,Y + \sum_{i=1}^{c} Z^{-i}\prod\left(-a_{c-i},\,X\right) \qquad (32)$$

In order to share the Z^{-c} operation, this can be transformed into:

$$\prod\left(a_c,\,X\right) = \sum_{i=1}^{c} Z^{-i}\left[\prod\left(-a_{c-i},\,X\right) + \epsilon_{i,c}Y\right] \qquad (33)$$

where $\epsilon_{i,c} = 0$ if $i \neq c$ and $\epsilon_{c,c} = 1$.

Since $a_c \neq 0$, X can be expressed explicitly by:

$$X = a_c^{-1}\sum_{i=1}^{c} Z^{-i}\left[\prod\left(-a_{c-i},\,X\right) + \epsilon_{i,c}\,Y\right] \qquad (34)$$

The network for performing this computation is shown in Figure (F16)

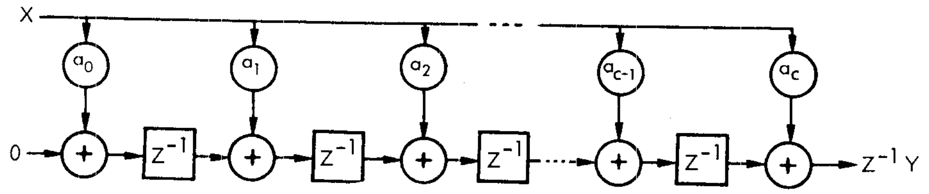

Figure (F13): Polynomials multiplication (most significant term leading).

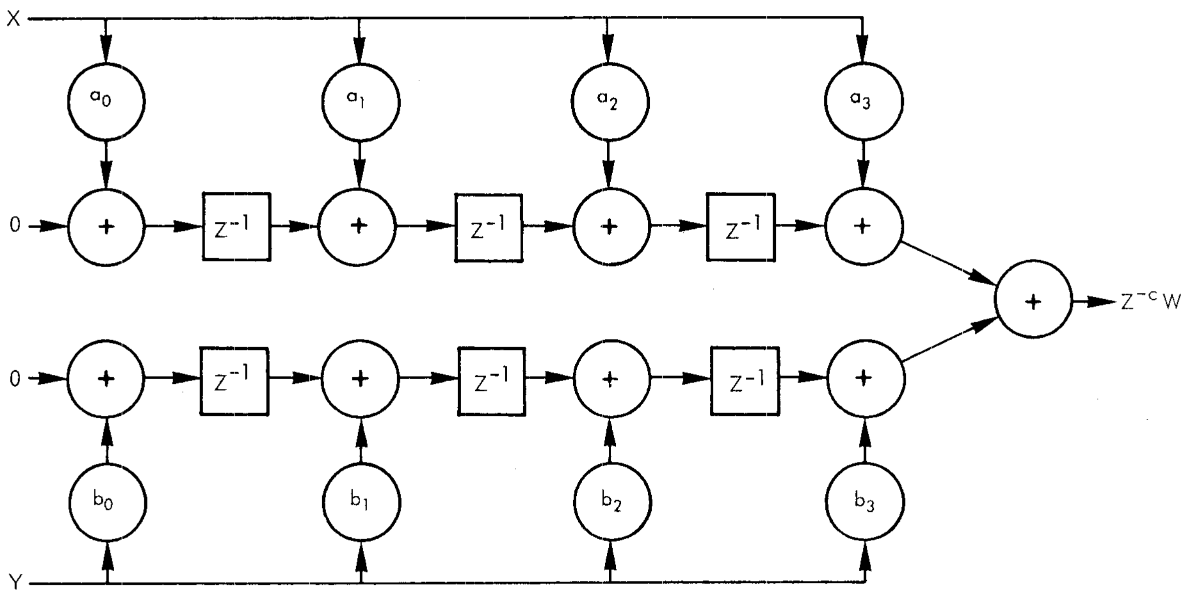

Figure (F14): Sum of polynomials products.

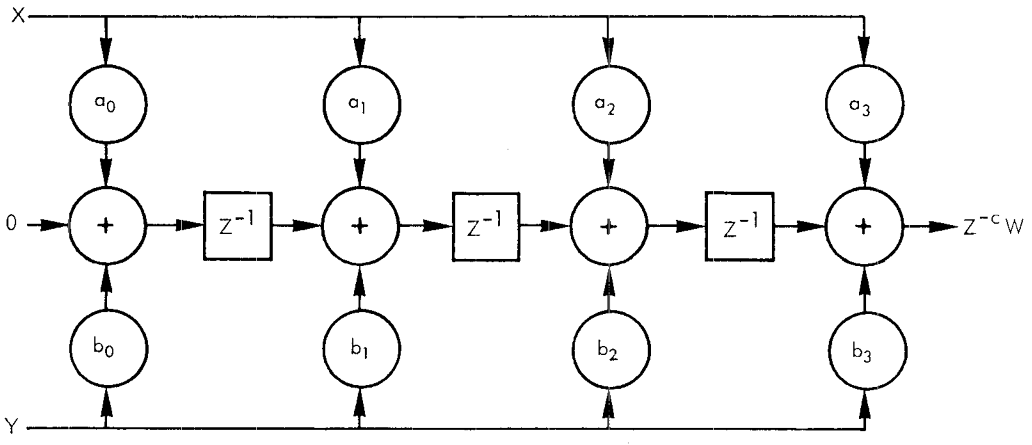

Figure (F15): Sum of polynomials products, combined.

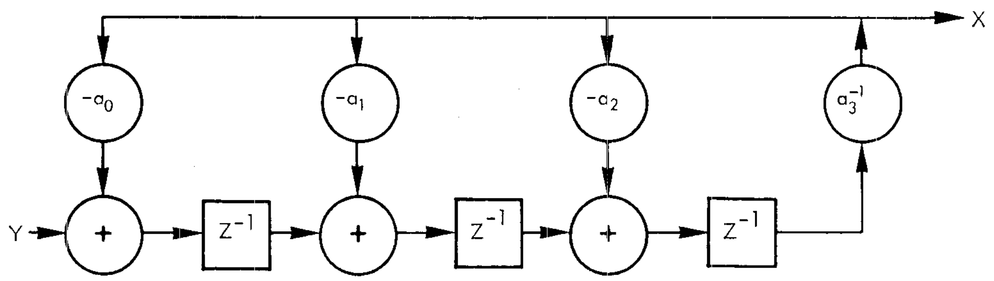

Figure (F16): Polynomial division, for c = 3

Since X is synchronized with Y, x_j is computed and is available at the same cycle when y_j is given. Since the first coefficient of Y is y_{m+c}, and the first coefficient of X is x_m, during the first c cycles no x_i is output.

Before starting this operation all the Z units are cleared. Then the Y coefficients are given, one at a time (i.e., one per cycle). The first c cycles are initialization cycles, and no output is expected. During the next $m+1$ cycles all the coefficients of X, with x_m leading and x_o trailing are available.

At this point the Z units include the same data which was present in the Z units of the network shown in Figure (F13), just before the multiplication process started.

Since all the Z units in this network were cleared before the multiplication, all the Z-units should contain zeroes after the division. If they are discovered to contain any non-zero value, then $Y(t)$ was not a product of $A(t)$ by any polynomial.

In fact, the values in the c delay-units are the coefficients of the remainder polynomial, $R(t)$, whose degree is less then m. This polynomial is defined by

$$R(t) = Y(t) - A(t)\,X(t) \qquad (35)$$

Checking the Multiplication and the Division

In order to check (which is weaker than "verify") these operations we prove that if we use these networks first to perform the multiplication of any arbitrary polynomial, $X(t)$, by the given polynomial, $A(t)$, and then to perform the division of this product by the same given polynomial, $A(t)$, then the same arbitrary polynomial $X(t)$, results.

Let $Y(t)$ be the result of the multiplication of $X(t)$ by $A(t)$, and let $S(t)$ be the result of the division of $Y(t)$ by $A(t)$. We will prove that $S(t) = X(t)$.

From (32)

$$S = a_c^{-1}\left[Z^{-c}\,Y + \sum_{i=1}^{c} Z^{-i} \prod \left(-a_{c-i}, S\right) \right] =$$

substitute (26)

$$= a_c^{-1}\left[\sum_{i=0}^{c} Z^{-i} \prod \left(a_{c-i}, X\right) - \sum_{i=1}^{c} Z^{-i} \prod \left(a_{c-i}, S\right) \right] =$$

$$= a_c^{-1}\left[a_c\,X + \sum_{i=1}^{c} Z^{-i} \prod \left(a_{c-i}, X\right) - \sum_{i=1}^{c} Z^{-i} \prod \left(a_{c-i}, S\right)\right] =$$

$$= X + a_c^{-1} \sum_{i=1}^{c} Z^{-i} \prod \left(a_{c-i}, X-S\right) =$$

$$= S + a_c^{-1}\, a_c \left(X-S \right) + a_c^{-1} \sum_{i=1}^{c} Z^{-i} \prod \left(a_{c-i}, X-S\right) =$$

$$= S + a_c^{-1} \sum_{i=0}^{c} Z^{-i} \prod \left(a_{c-i}, X-S\right) \qquad (36)$$

Hence

$$\sum_{i=0}^{c} Z^{-i} \prod \left(a_{c-i}, X-S\right) = 0 \qquad (37)$$

Since the polynomial $A(t)$ is known not to be the zero polynomial because $a_c \neq 0$, the polynomial $S(t)$ must be equal to $X(t)$. *Q.E.D.*

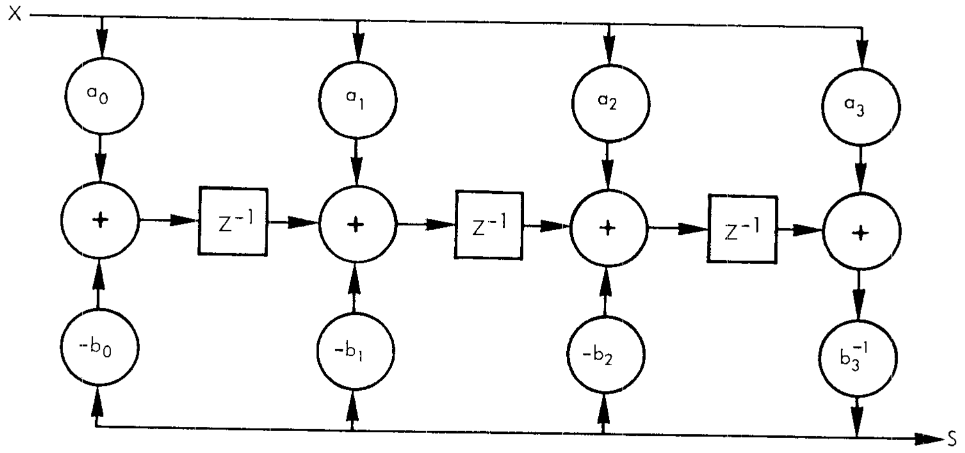

Figure (F17): The $S = (A\ X) \,/\, B$ implementation, for $c = 3$.

Simultaneous Multiplication and Division of Polynomials

Define $S(t)$ to be the polynomial which is obtained by multiplying the arbitrary polynomial $X(t)$ by the given polynomial $A(t)$, and then by dividing this product by another given polynomial, $B(t)$, also of degree c, such that $b_c \neq 0$.

By following (36) we get

$$S = b_c^{-1} \left[\sum_{i=0}^{c} z^{-i} \prod \left(a_{c-i}, X\right) + \right.$$

$$\left. + \sum_{i=1}^{c} z^{-i} \prod \left(-b_{c-i}, S\right) \right] =$$

$$= b_c^{-1} \left\{ a_c X + \sum_{i=1}^{c} z^{-i} \left[\prod \left(a_{c-i}, X\right) + \right.\right.$$

$$\left.\left. + \prod \left(-b_{c-i}, S\right) \right] \right\} \qquad (38)$$

The network which performs this computation is shown in figure (F17).

6. SUMMARY AND CONCLUSIONS

We showed that the mathematical notation which is commonly used for the specification of a computation may implicitly suggest some design features which are not necessarily desired.

We suggest that the mathematical definition be transformed into the computational network representation notation, which can be evaluated according to the important design objectives.

Furthermore, this representation can be symbolically transformed in order to generate other alternative networks, which should also be evaluated according to the design objectives.

These transformations should continue until no further improvement is achieved.

Furthermore, we suggest that it is feasible to implement an automatic system for performing these symbolic transformations and evaluations, and highly recommend it.

7. ACKNOWLEDGMENTS

I am grateful to Bob Sproull (of CMU) for encouraging me to write this note, to Bob Kahn (ARPA) for suggesting the application of this technique to polynomial multiplication and division, and to Chuck Seitz (Caltech) for his critical review of the draft of this paper.

I would like to thank Ms. Lisa Moses for her skillful typing of this note, and especially the mathematical formulae. I would like to thank Ms. Debe Hays for her help, and Mr. Nelson Lucas for his help in preparing the many drawings which are an integral part of this paper.