NUMERICAL LIMITATIONS ON THE DESIGN OF DIGIT ONLINE NETWORKS*

Robert Michael Owens
Mary Jane Irwin

Department of Computer Science
The Pennsylvania State University
University Park, PA 16802 USA

Abstract

A fully digit online arithmetic unit generates at least the i most (least) significant digits of the result after having been supplied no more than the $(i+k)$ most (least) significant digits of each operand, where k is a small constant. This digit serial property can be used to reduce the aggregate fill and flush times of a chained array of digit online arithmetic units and to reduce their VLSI interconnection complexity. However, because of this digit serial property, unique and inherent limitations may have to be imposed on any arithmetic unit which performs digit online operations. For some calculations, these limitations may be so severe as to make digit online evaluation virtually impossible. We show several important signal processing problems where these limitations have either been avoided or their effect greatly reduced.

Introduction

Digit online arithmetic units have a digit serial property which distinguishes them from more conventional full precision arithmetic units. This digit serial property permits the chaining of the digits of the output(s) of one digit online unit directly into the input(s) of another. As a result, networks of digit online arithmetic units offer the promise of being able to solve certain types of numerical problems significantly faster than other more conventional networks of comparable hardware complexity. Unfortunately, digit serial processing of floating point numbers requires that several unique limitations be imposed on the operands. If the effect of these limitations can not be resolved, then it may not be possible to build digit online networks to accurately solve important problems which can be solved by more conventional networks. Since these limitations do not exist, or have a negligible effect in conventional networks, they have not received wide exposure to investigation. We will show several important signal processing problems where these limitations have either been avoided or their effect greatly reduced. Hence, the speed and the implementation advantages of digit online networks can be used to solve these problems.

* This research was supported in part by the National Science Foundation (MCS-7809126A01) and the Office of Naval Research (N00014-80-C-0517).

Definitions and Notations

First some definitions and notations are presented. A nonredundant, unsigned, fixed point number system F can be concisely characterized by the integer values r, p, and s, which are, respectively, the radix, precision, and scale factor, and by a set D of symbols $\{d_0, d_1, \ldots d_n\}$ called digits. Such a number system will be designated by $F(r, p, s, D)$. Each digit d_i, $1 \le i \le n$, has a predefined integer value associated with it. The *representation set* R_F of the fixed point number system F is a set of representations each of which has the form:

$$d_{i_1} d_{i_2} \ldots d_{i_p} .$$

The value associated with any representation x in F is $\sum_{j=1}^{p} d_{i_j} r^{(s-j)}$. The digit $(x)_j = d_{i_j}$ is the j'th digit of the representation. The digits $d_{i_1}, d_{i_2}, \ldots, d_{i_k}$ and the k *most significant digits* of x, while the digits $d_{i_{p-k+1}}, d_{i_{p-k+2}}, \ldots, d_{i_p}$ are the k *least significant digits* of x. Conventional fixed point number systems are nonredundant since the representation associated with each value is distinct. Redundant systems, on the other hand, have several different representations corresponding to the same value. Let $V_F = \{v^i, 1 \le i \le m\}$ be the set of numerical values which can be exactly represented in F. Suppose the set V_F is ordered such that

$$i < j \Rightarrow v^i < v^j, \quad 1 \le i \le m, \quad 1 \le j \le m .$$

Then the fixed point number system F is *uniformly distributed*, if there exists a constant v_e such that

$$v_e = v^{i+1} - v^i, \quad 1 \le i \le m - 1 .$$

If $s = p$, F is an integer number system $(v_e = 1)$ and, if $s = 0$, F is a fractional number system $(v_e = r^{(-p)})$.

Consider performing certain fixed point operations in a digit online manner. If F is some arbitrary fixed point number system and O is some operator, then, $O(x^1, x^2, \ldots, x^n)$, $x^j \in F$, $1 \le j \le n$, is performed in a *right directed* digit online manner if at least the i most significant digits of the result are generated after no more than the $(i+k)$ most significant digits of each input x^j have been consumed. The operation is performed in a *left directed* digit online manner if at least the i least significant digits of the result are generated after no more than the $(i+k)$ least significant digits of each input x^j have been consumed. The small, predefine constant k corresponds to the *digit online delay*. The drawback with left directed algorithms is that the set of arithmetic operations for which there exist a left directed digit online algorithm is quite small (basically integer addition and multiplication)[6]. We restrict our attention in this paper to right directed algorithms for which there exist a family of right directed digit online algorithms[4].

Unfortunately, several unique and inherent limitations must be dealt with when performing certain floating point digit online operations. For some operations, these limitations may be so severe as to make performing these operations in a digit online manner virtually impossible. For other operations, these limitations may not be nearly as severe; it may be possible either to avoid (in much the same way that overflow and underflow are avoided in a conventional arithmetic units) these limitations or to show (in much the same way error analysis for conventional arithmetic unit indicates that accurate answers are being generated) that the effects produced by the limitations are manageable. Since these limitations may interact, it is sometimes difficult to determine the overall impact of one particular limitation. These limitations are directly or indirectly related to the allowable operand range as detailed in the next section.

Digit Online Arithmetic

Let (x_c, x_m) and (y_c, y_m) be possibly unnormalized floating point numbers where x_c is the integer exponent and x_m is the fractional mantissa. The definition of the *exact* floating point sum (z_c, z_m) of (x_c, x_m) and (y_c, y_m) is given by:

$$(z_c, z_m) = (x_c, x_m) + (y_c, y_m) =$$
$$(max(x_c, y_c) + u_c, (x_m r^{(x_c \dot{-} y_c)} + y_m r^{(y_c \dot{-} x_c)}) u_m) =$$
$$(max(x_c + u_c, y_c + u_c), x_m r^{(x_c - z_c)} + y_m r^{(y_c - z_c)}),$$

where u_c is any integer, $u_m = r^{(-u_c)}$ and

$$x \dot{-} y = \begin{cases} x - y & if \ x \ge y \\ 0 & otherwise \end{cases}$$

For example consider producing the sum of (-14.,+.4723) and (-14.,+.6435). In this case the following result is obtained

$$(z_c, z_m) = (-14., +.4723) + (-14., +6435) =$$
$$(max(-14., -14.) + u_c, (+.4723 r^{(-14. \dot{-} -14.)} +$$
$$+.6435 r^{(-14. \dot{-} -14.)}) u_m) =$$
$$(-14. + u_c, +1.1158 u_m).$$

Note that if $u_c = 1$ then the result is (-13.,+.11158) and overflow of the mantissa has been prevented. Hence, mantissa overflow can be prevented by a judicious choice of u_c. Now, consider producing the sum of (+12.,+.1764) and (+12.,-.1723). In this case the following result is obtained:

$$(z_c, z_m) = (+12., +.1764) + (+12., -.1723) =$$
$$(max(+12., +12.) + u_c, (+.1764 r^{(+12. \dot{-} +12.)} +$$
$$-.1723 r^{(+12. \dot{-} +12.)}) u_m) =$$
$$(+12. + u_c, +.0041 u_m)$$

Note that if $u_c = -2$, then the results is the normalized value (+10.,+.4100). Hence, unnormalized mantissas can also be prevented by a judicious choice of u_c.

It would be most desirable to be able to adjust u_c anywhere in the range $+1$ to $-p$. Unfortunately, in digit online arithmetic such a luxury is not possible. The digit online delay will increase correspondingly when the choice for u_c is other than 1 or 0 (subtracting one from the lower limit adds one to the online delay). Hence, to keep a small digit online delay, the lower bound on u_c must, in turn, be kept close to zero. Only small digit online delays permit effective chaining of digits between digit online arithmetic units and are, thus, directly related to gains in processing speed[2,3].

Digit online arithmetic units do not escape the problem also encountered in conventional arithmetic units of having to deal with rounding of results before storage. Consider the result (-13.,+.11158) in the first example above. To store this result when $p = 4$, it must be rounded to some nearby representable value such as (-13.,+.1116).

With these two considerations in mind, an algorithm[4] has been formulated to perform digit online floating point addition. The definition for the *computed* digit online floating point sum $(\hat{z}_c, \hat{z}_m)$ of (x_c, x_m) and (y_c, y_m) as produced by this algorithm is given by:

$$(\hat{z}_c, \hat{z}_m) = (x_c, x_m) \hat{+} (y_c, y_m) =$$
$$(max(x_c, y_c) + \hat{u}_c, (x_m r^{(x_c \dot{-} y_c)} + y_m r^{(y_c \dot{-} x_c)}) \hat{u}_m + e) =$$
$$(max(x_c + \hat{u}_c, y_c + \hat{u}_c), x_m r^{(x_c - \hat{z}_c)} + y_m r^{(y_c - \hat{z}_c)} + e),$$

where

$$\hat{u}_c = \begin{cases} 1 & if \ | x_m r^{(x_c \dot{-} y_c)} + y_m r^{(y_c \dot{-} x_c)} | \ge h \\ 0 & if \ | x_m r^{(x_c \dot{-} y_c)} + y_m r^{(y_c \dot{-} x_c)} | \le \tilde{h} \\ 1 \, or \, 0 & otherwise \end{cases}$$

$\hat{u}_m = r^{(-\hat{u}_c)}$, and e is some constant such that $|e| \le v_e$. The constants h and $\tilde{h}$ are such that

$1 \ge h \ge \tilde{h} \ge r^{(-1)}$. The ambiguity in the definition of $\hat{u}_c$ is not important for the purposes of this paper. Without loss of generality it can be assumed that $u_c = \hat{u}_c$. Hence, $(z_c, z_m) = (\hat{z}_c, \hat{z}_m + e)$.

Note that in the definition of the computed digit online floating point sum, shifting of the value formed by the sum of the aligned mantissas is limited to at most a right shift of one digit. This is done to handle mantissa overflow. Further, since the mantissa is never left shifted, there can be no attempt at normalization of the result's mantissa. However, never allowing left shifts minimizes the online delay.

Again, let (x_c, x_m) and (y_c, y_m) be possibly unnormalized floating point numbers. The definition for the *exact* floating point product (z_c, z_m) of (x_c, x_m) and (y_c, y_m) is given by

$$(z_c, z_m) = (x_c, x_m) * (y_c, y_m) =$$
$$(x_c + y_c + u_c, x_m y_m u_m) =$$
$$(x_c + y_c + u_c, x_m y_m r^{(x_c + y_c - z_c)}),$$

where u_c is any integer and $u_m = r^{(-u_c)}$. Overflow of the mantissa is not possible in multiplication. But, consider multiplying (+12.,+.0234) and (+14.,-.0534) to obtain:

$$(z_c, z_m) = (+12.,+.0234) * (+14.,-.0534) =$$
$$(+26.+u_c, -.00124956u_m)$$

Note that, if $u_c = -2$, then the result is the normalized value (+24.,-.124956). Hence, unnormalized mantissas can once again be prevented by a judicious choice of u_c.

As in addition, it would be most desirable to be able to adjust u_c, in this case anywhere in the range of 0 to $-p$. But again a lower limit close to zero must be placed on u_c to keep the digit online delay at a minimum. Also, as in addition, the problem of rounding must be addressed.

With these two considerations in mind, an algorithm[4] has been formulated to perform digit online floating point multiplication. The definition for the *computed* digit online floating point product $(\hat{z}_c, \hat{z}_m)$ of (x_c, x_m) and (y_c, y_m) as produced by this algorithm is given by:

$$(\hat{z}_c, \hat{z}_m) = (x_c, x_m) \hat{*} (y_c, y_m) =$$
$$(x_c + y_c + \hat{u}_c, x_m y_m \hat{u}_m + e) =$$
$$(x_c + y_c + \hat{u}_c, x_m y_m r^{(x_c + y_c - \hat{z}_c)} + e),$$

where

$$\hat{u}_c = \begin{cases} -1 & if\ |x_m y_m| \le \tilde{h}/r \\ 0 & if\ |x_m y_m| \ge h/r \\ -1\ or\ 0 & otherwise \end{cases}$$

$\hat{u}_m = r^{(-\hat{u}_c)}$, and e is some constant such that $|e| \le v_e$. The constants h and $\tilde{h}$ have been previously defined. The ambiguity in the definition of $\hat{u}_c$ is again not important for the purposes of this paper. Without loss of generality it can be assumed that $u_c = \hat{u}_c$. Hence, $(z_c, z_m) = (\hat{z}_c, \hat{z}_m + e)$.

Note that in the definition of the computed digit online floating point product, shifting of the value formed by the product of the mantissas is limited to at most a left shift of one digit. Hence, very little is done to try to normalize an unnormalized result. Right shifting is not necessary since overflow can not occur.

An interesting question remains as to the effects this use of unnormalized arithmetic has on the suitability of solving problems using digit online arithmetic units. We investigate these effects in the next section.

Numerical Problems

The first problem to be considered is the extended summation given by:

$$s^i = \begin{cases} b^i & if\ \ i = 1 \\ b^i + s^{i-1} & if\ \ 2 \le i \le n \end{cases}$$

In this case, it has been shown[5], that under a fairly easy to meet set of conditions, $(\hat{s}^n_c, \hat{s}^n_m)$ equals the exact extended floating point sum of $(b^i_c, b^i_m + \hat{e}^i)$, $1 \le i \le n$, where the $\hat{e}^i$, $1 \le i \le n$, are constants such that $|\hat{e}| \le \frac{2nrv_e}{\tilde{h}}$. That is, the error in the computed digit online extended floating point sum could also be obtained by computing the exact result where a value no larger than $\frac{2nrv_e}{\tilde{h}}$ has been added to each operand. Hence, it is said that log n digits of accuracy are lost when producing the computed result. Note that the backward error bound does not depend on the degree to which the operands are unnormalized and, furthermore, that unless some special ordering information is known about the b^i's this bound is as good as (within a constant factor) the bound which would be obtained even if normalized arithmetic were used[8]. Hence, this problem is as suitable for computation by a digit online arithmetic unit as it is by a conventional arithmetic unit.

The second problem to be considered is the extended product given by:

$$p^i = \begin{cases} a^i & i = 1 \\ a^i * p^{i-1} & for\ \ 2 \le i \le n \end{cases}$$

Let k^i, $1 \le i \le n$, be integer constants such that

$$r^{(-1)} \le |a^i_m|\ r^{(k^i)} < 1$$

The k^i's are nothing more than a "normalization shift count". The existence of the constants k^i, $1 \le i \le n$, implies that $a^i_m \ne 0$. Furthermore, since $|a^i_m| < 1$, $1 \le i \le n$, $k^i \ge 0$. In this case, it has been shown[5] that under a fairly easy to meet set of conditions that $(\hat{p}_c^n, \hat{p}_m^n)$ equals the exact extended floating point product of $(a_c^i, a_m^i + \hat{e}^i r^{(k^{1,i-1}+1)})$, $1 \le i \le n$, where $k^{1,i-1} = \sum_{j=1}^{i-1} k^j$, and the $\hat{e}^i$, $1 \le i \le n$, are constants such that $|\hat{e}^i| \le \frac{4rv_e}{\tilde{h}}$.

Note that, unlike the case with summation, the backward error bound in multiplication does depend on the degree to which the floating point numbers (a_c^i, a_m^i), $1 \le i \le n-1$, are unnormalized. Also the amount of unnormalization which can be tolerated is greatly restricted. In fact, if

$$\left| \sum_{i=1}^{n} \hat{e}^i r^{(k^{1,i-1}+1)} \right| \ge 1$$

then the magnitude of the error in the computed digit online extended floating point product is greater than the magnitude of the exact extended floating point product itself. However, if the a^i's are normalized then the bound, somewhat as expected, is as good as the bound which would be obtained if normalized arithmetic were used. Hence, if the a^i's are normalized, this problem is as suitable for computation by a digit online arithmetic unit as it is by a conventional arithmetic unit.

The third problem to be considered is a simple first order linear recurrence given by:

$$x^i = \begin{cases} b^i & i = 1 \\ b^i + a^i x^{i-1} & for \quad 2 \le i \le n \end{cases}$$

In this case, it has been shown[5] that if the a^i's are normalized and if certain other easy to meet conditions are satisfied, that $(\hat{x}_c^n, \hat{x}_m^n)$ equals the exact floating point recurrent of $(b_c^i, b_m^i + \hat{e}^i)$, $1 \le i \le n$, and (a_c^i, a_m^i), $2 \le i \le n$, where $\hat{e}^i$, $1 \le i \le n$, are constants such that $|\hat{e}^i| \le \frac{4nrv_e}{\tilde{h}}$. While it is possible to relax the assumption that the a^i's are normalized (the effect is similar to that in multiplication), there is, in practice, little motivation in doing so. Note that under the assumption that the a^i's are normalized, the bound on the backward error is a good as the error which would be obtained even if normalized arithmetic were used. Hence, if the a^i's are normalized, this problem is as suitable for computation by digit online arithmetic unit as it is by a conventional arithmetic unit.

The fourth problem to be considered is the inner product given by:

$$x^i = \begin{cases} a^i b^i & i = 1 \\ a^i b^i + x^{i-1} & for \quad 2 \le i \le n \end{cases}$$

Note that if a^i's and the b^i's are normalized, then the products $a^i b^i$, $1 \le i \le n$ (as produced by digit online arithmetic units) are normalized or very nearly so (the mantissas of the products are greater than or equal to $\tilde{h}$). An extension of the result for the extended sum problem shows that in this case, the bound on the backward error increases by only a multiplicative factor of $\frac{1}{\tilde{h}}$. Hence, this problem is as suitable for computation by a digit online arithmetic unit as it is by a conventional arithmetic unit if the operands are normalized.

The previous four problems lead to several observations. Note that the inner product operation can be performed as accurately by a digit online arithmetic unit as by a conventional arithmetic unit. Since the inner product operation is basic to several methods used to solve many vector-vector and matrix-vector problems, these problems can be solved by digit online arithmetic units if the input data is normalized and an appropriate algorithm is used. The following section discusses two signal processing problems which can be solved using digit online arithmetic units.

Networks for Digital Signal Processing

Most of the major computational requirements for many important real-time signal processing tasks can be reduced, perhaps after some effort, to matrix computations. These include matrix-vector multiplication, matrix-matrix multiplications and additions, matrix inversions, the solution of linear systems, eigensystem solutions, etc. The problems are made interesting, from the point of view of computer designers, by the demands of real-time computation and the sheer volume of input. Two digit online networks, one for digital filtering and one for discrete Fourier transform, are proposed in this section. Each is suited to VLSI implementation on a chip or several interconnected chips[2]. The argument is made, based on the results from the previous section, that these networks produce as accurate results as conventional networks but with improved speed.

First consider solving the discrete Fourier transform. An n-pont discrete Fourier transform (DFT) problem can be reduced to the following matrix-vector multiplication:

$$x^i = \sum_{j=0}^{n-1} a^j \omega^{(ij)}$$

where ω is and nth root of unity and $a^0, a^1, \ldots, a^{n-1}$ are given constants. A linear digit online network for solving the DFT recursion can be defined as in Figure 1. The inputs are provided to the network in a digit serial, most significant digit first mode. As the digits are being input, digit online processing is also in progress. As a specific illustration, consider a DFT for the case of $n = 5$. The five-point DFT consists of evaluating the polynomial

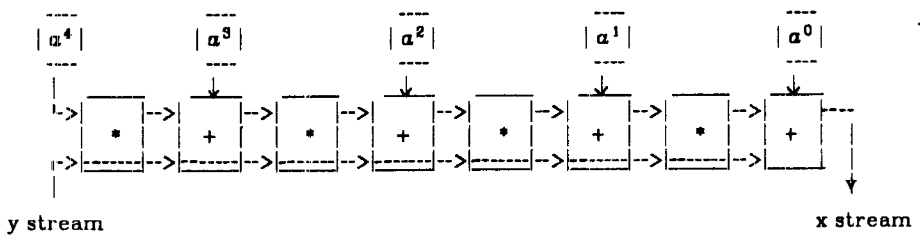
$$a^4 y^{(4)} + a^3 y^{(3)} + a^2 y^{(2)} + a^1 y + a^0$$

at $y = 1$, ω, $\omega^{(2)}$, $\omega^{(3)}$, and $\omega^{(4)}$ where ω is a fifth root of unity. Horner's rule evaluation of this polynomial gives

$x^0 = (((a^4 * 1 + a^3) * 1 + a^2) * 1 + a^1) * 1 + a^0$

$x^1 = (((a^4 * \omega + a^3) * \omega + a^2) * \omega + a^1) * \omega + a^0$

$x^2 = (((a^4 * \omega^{(2)} + a^3) * \omega^{(2)} + a^2) * \omega^{(2)} + a^1) * \omega^{(2)} + a^0$



where the y stream input is $y^0 = 1$, $y^1 = \omega$, $y^2 = \omega^{(2)}$, $y^3 = \omega^{(3)}$, and $y^4 = \omega^{(4)}$ and the x stream output is x^0, x^1, x^2, x^3, and x^4.

Figure 1. Linear Digit Online DFT Network

$$x^3=(((a^4*\omega^{(3)}+a^3)*\omega^{(3)}+a^2)*\omega^{(3)}+a^1)*\omega^{(3)}+a^0$$
$$x^4=(((a^4*\omega^{(4)}+a^3)*\omega^{(4)}+a^2)*\omega^{(4)}+a^1)*\omega^{(4)}+a^0$$

The x^i's can be computed by a one-dimensional, linear digit online network, shown in Figure 1.

Each of the processing cells in Figure 1 is either a digit online adder or multiplier. The top output line is the product or sum of the two inputs to the cell, while the lower input is routed through to the lower output. The indicated registers are digit-by-digit circular shift registers which have been preloaded with the appropriate constants. The y stream is input one digit at a time, most significant digit first, and the x stream is output one digit at a time, most significant digit first, after a start-up delay of $(2c + k)t_d$ where c is the number of digit online processors or cells, k is the online delay, and t_d is the processing time for each cell. This is assuming that the digit online delay of all of the cells in the network are the same. If the digit online delays of the cells are not all the same, buffering must be provided between adjacent cells

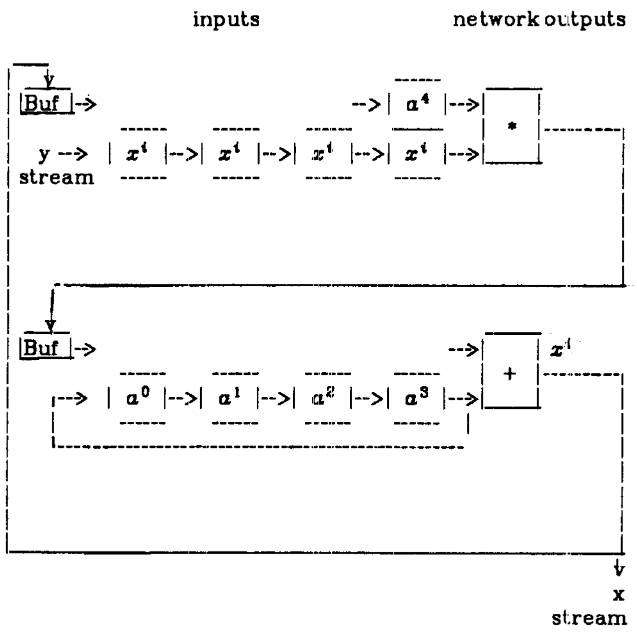


Figure 2. Recursive Digit Online DFT Network

and the maximum k used in the above timing equation. The indicated digits stream between the interconnected processors allowing concurrent operations by all processors. Since the DFT recursion can be reduced to the inner product problem in the previous section, the digit online network for solving the DFT will give as accurate a result as the more conventional network with a significant speed improvement.

Yet another digit online configuration for solving the five-point DFT recurrence is shown in Figure 2 using only two digit online processors. The shift registers in Figure 2 supply one digit starting at the most significant end of the appropriate operand at the beginning of each time step. They are initially loaded with a^4, a^3 through a^0, and four copies of $y^i = y^0 = 1$. The y stream consists of four copies of each of y^1 through y^4 which are shifted into the registers feeding the digit online multiplier along with the appropriate digits of x^i. With this configuration, $O(n)$ processing time can still be maintained even with only two digit online processors, as long as a sufficient number of shift registers are provided.

Another frequently performed signal processing computation is that of digital filtering. The filtering problem can also be reduced to that of matrix-vector multiplications. The Infinite Impulse Response (IIR) filter can be described by the following recurrence.

$$y^i = \sum_{j=0}^{h} w^j x^{i-j} + \sum_{j=1}^{k} r^j y^{i-j}$$

where $w^0, w^1, \ldots w^h$ and $r^1, r^2, \ldots, r^k$ are weighting coefficients,

$y^{-k}, y^{-k+1}, \ldots, y^{-1}$ are initial values, and

$x^{-h}, x^{-h+1}, \ldots, x^0, x^1, \ldots, x^n$ is the input sequence.

Each summation in the above recurrences has the form of a Finite Impulse Response filter (FIR).

A digit online array network can also be defined for the filtering problem, but with a drastically different configuration from the previously considered networks. Consider the case when $h = k = 2$. The array must be designed to evaluate the following

$$y^0 = w^0x^0 + w^1x^{-1} + w^2x^{-2} + r^1y^{-1} + r^2y^{-2}$$
$$y^1 = w^0x^1 + w^1x^0 + w^2x^{-1} + r^1y^0 + r^2y^{-1}$$

$y^2 = w^0x^2 + w^1x^1 + w^2x^0 + r^1y^1 + r^2y^0$

The y^i's can be computed by a digit online array network as shown in Figure 3.

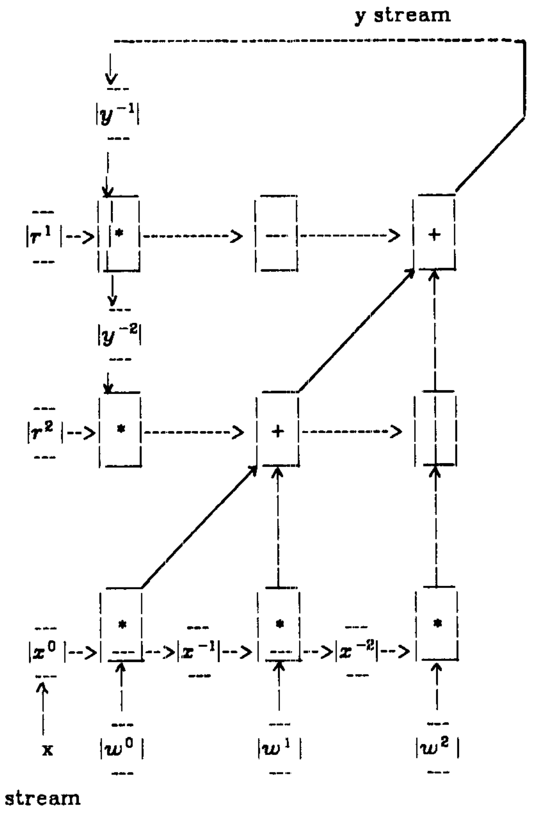


Figure 3. Array Digit Online Filter Network

The interconnected shift registers in Figure 3 are preloaded as indicated and then supply one digit of the appropriate operand at the beginning of each cycle. The input x stream must match the output pace of the y stream, since the generated y values are reused by the network. Once again, buffering would be necessary if the digit online delays of the processors were not the same. Such a network would satisfy the VLSI constraints of a few types of relatively simple cells, simple and regular data and control flow, concurrent use of the cells in the network to provide a high computation rate, and multiple use of each input data item to reduce the demands on the bandwidth between network and host. Since the filtering recursion can also be reduced to the inner product problem in the previous section, the digit online network for solving it will give as accurate a result as the more conventional network with a significant speed improvement.

Conclusions

We have shown that there exist methods by which some vector-vector and vector-matrix problems can be solved as accurately by a digit online network as they can by a more conventional network. Furthermore, the properties which made the digit online network preferable to the more conventional network are retained. It would be nice if we could be as conclusive or positive about all such algorithms. But this is not the case. Consider, for example the problem given by:

$$s^i = \begin{cases} b^i & \text{if } i = 1 \\ s^{i-1} * (b^i + a^i) & \text{if } 2 \le i \le n \end{cases}$$

Note that the result of each of the sums $b^i + a^i$, $1 \le i \le n$ may not necessarily be normalized. Hence, it would be disastrous to try to form their product using digit online operations.

An interesting open question is whether the FFT can be solved as accurately by a digit online network as it can by a more conventional network. Since the FFT is central to many algorithms used to solve some vector-matrix problems, it would indeed be most rewarding if it were.

References

1 Ercegovac, M.D. and A.L. Grnarov, "On the Performance of On-Line Arithmetic", *Proceedings of the 1980 International Conferences on Parallel Processing*, pp. 55-62, August 1980.

2 Irwin M.J., "Some Applications of Digit Online Networks to Signal Processing", Report CS-82-14, PSU, July 1982.

3 Irwin, M.J. and Owens, R.M., "Fully Digit Online Networks", *IEEE Transactions on Computers*, April 1983.

4 Owens, R.M., "Digit On-line Algorithms for Pipeline Architectures", Ph.D. Thesis, Report CS-80-21, Department of Computer Science, The Pennsylvania State University, University Park, PA, August 1980.

5 Owens, R.M., "Error Analysis of Unnormalized Arithmetic", Report CS-81-16, Department of Computer Science, The Pennsylvania State University, University Park, PA, August 1981.

6 Owens, R.M., "Techniques to Reduce the Inherent Limitations of Fully Digit Online Arithmetic", *IEEE Transactions on Computers*, April 1983.

7 Watanuki O. and M.D. Ercegovac, "Floating Point On-line Arithmetic: Algorithms and Error Analysis", *Proc. of the 5th Computer Arithmetic*, pp. 81-91, May 1981.

8 Wilkinson, J.H., *Rounding Errors in Algebraic Processes*, Prentice Hall, Inc., 1963.