AREA-TIME OPTIMAL VLSI NETWORKS FOR COMPUTING INTEGER MULTIPLICATION AND DISCRETE FOURIER TRANSFORM

F. P. Preparata[1] and J. E. Vuillemin[2]

Abstract

In this paper we present a class of VLSI networks for computing the Discrete Fourier Transform and the product of two N-bit integers. These networks match, within a constant factor, the known theoretical lower-bound $O(N^2)$ to the area $\times$ $(\text{time})^2$ measure of complexity. While this paper's contribution is mainly theoretical, it points toward very practical directions: we show how to design multipliers with area $A = O(N)$ and time $T = O(\sqrt{N})$ on one hand, and $A = O((N/\log^2 N)^2)$, $T = O(\log^2 N)$ on the other. Both of these designs should be contrasted with the currently available multipliers, whose performances are $A = O(N)$, $T = O(N)$ or even $A = O(N^2)$, $T = O(N)$.

1. Introduction

Very-Large-Scale-Integration (VLSI) is having a profound impact on the design methodology of digital systems. One of the basic reasons is that VLSI integrates processing elements and their interconnection in the same medium, the silicon chip. Thus, the traditional component count criterion is superseded by the notion of chip area, which reflects the complexity both of computation and of data communication.

The current approach to the evaluation of the performance of a VLSI design for the solution of a given problem is based on the computational model of VLSI, which has emerged through the contributions of several authors, primarily Mead-Conway [1] and Thompson [2,3], and subsequently Brent-Kung [4] and Vuillemin [5]. In this model, a circuit is a graph whose nodes are gate and I/O ports, connected by wires. A wire has width λ (dependent upon the fabrication process), two wires can cross only at a finite number of points, and at most ν wires overlap at any point. The areas of a gate and of a port are λ^2 and $C\lambda^2$, respectively, and each input bit is available just once. The total circuit area A is due to wires, gates, ports, and empty spaces. Computation is synchronous and the combined gate-computation and

[1] F. P. Preparata is with the Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801. His work was supported in part by the National Science Foundation under Grant MCS 78-13642 and the Joint Services Electronics Program under Contract N00014-79-C-0424.

[2] J. E. Vuillemin is with the Laboratoire de Recherche en Informatique, Université de Paris-Sud, 91405 Orsay, France. His work was supported in part by ERA 452 "al Khowarizmi" of the C.N.R.S.

result-transmission (on a wire between two nodes) takes some time τ, technology-dependent. The total computation T is the time elapsed between the beginning and the end of the chip computation. On the basis of arguments on the information transfer inside the VLSI chip of area A realizing a given algorithm, natural measures of complexity are the area $\times$ time products AT^2 and AT, or, in general, $AT^{2\alpha}$ for $0 \leq \alpha \leq 1$.

Several lower-bounds and upper-bounds to such measures have been obtained — for a variety of problems — by several workers, such as Thompson [2,3], Abelson and Andreae [6], Kung and Leiserson [7], Guibas et al. [8], Savage [9], Preparata-Vuillemin [10,11,12], and Brent and Kung [4].

Abelson-Andreae [6] and Brent-Kung [4], in particular, considered the problem of integer multiplication and independently showed that any VLSI circuit which multiplies two (N/2)-bit integers must have an $AT^{2\alpha}$ product $\Omega(N^{1+\alpha})$. Brent and Kung also proposed a multiplier network achieving $AT^{2\alpha} = O(N^{1+\alpha}(\log N)^{1+2\alpha})$.

In this paper we propose a class of area-time optimal designs for multiplier for all values of computation time T such that $O(\log^2 N) \leq T \leq O(\sqrt{N})$. Although the proposed networks may appear considerably complex for the current state of technology, they are the only area-time optimal ones known today, and may stimulate further research into more practical realizations. Our multipliers are heavily based on a DFT network which we also propose and which realizes the lower-bound $AT^2 = \Omega(N^2(\log N)^2)$ obtained by Thompson [3].

The paper is organized as follows. In Section 2 we discuss a general paradigm of recursive network, which is closely related to the cube-connected-cycles [10]. This paradigm is specialized in Section 3 to obtain an optimal FFT network, and in turn the latter is adapted in Section 4 to obtain the optimal multiplier. Although this paper could be shortened by merely describing our circuits for DFT and integer products, the discovery of such specific circuits arose in a more general framework. For the sake of a better understanding of these two constructs, we devote Section 2 to presenting circuits which are able to solve optimally a much wider class of problems, with applications besides DFT and integer product, as shown in [10].

2. General network scheme

The general scheme of the circuits to be described in this section is a network rendition of recursion. Specifically, suppose we have a problem where inputs and outputs are each n-component vectors of data items (for n even), which is solved by a recursive procedure, called ALGORITHM(0,n-1), of the following type (the input is initially stored in an array T[0:n-1], which after the execution of ALGORITHM will contain the output):

```
proc    ALGORITHM(0,n-1)
begin   ALGORITHM(0,n/2-1), ALGORITHM(n/2,n-1);
        foreach i : 0 ≤ i < n/2  par do (T[i],T[i+n/2]) ← OPER_i(T[i],T[i+n/2])
end
```

Here $OPER_i$ is an operation which updates the values of T[i] and T[i+n/2]. Although very simple, it has been shown in [10] that ALGORITHM is the paradigm of a large class of very important computations, such as merging, FFT, permutations, convolution, etc.

It is immediate that ALGORITHM naturally lends itself to implementation by a network as illustrated in figure 1. The circles denote processing modules, and the lines shown, called wires, are usable for the transfer of data items.[1] Moreover, for each pair of modules connected by a horizontal wire, we assign the computation capabilities to the left module, while the right module has a passive role and simply transmits an operand and receives a result. The generic left module, containing operand T[i], performs the following sequence of operations:

(i) it receives T[i+n/2] from its right companion;

(ii) it performs $OPER_i$ on the pair (T[i],T[i+n/2]) and obtains as a result the the ordered pair (T_L,T_R);

(iii) it stores T_L and transmits T_R to its right companion.

The "recursive boxes" are devoted to the execution of the recursive calls on half-size vectors, while the horizontal wires provide the interconnections necessary for OPER.

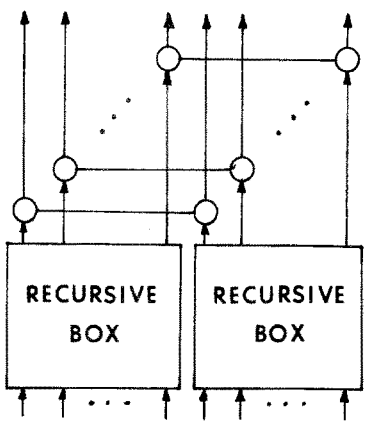

Figure 1. Basic network scheme.

[1] A wire may physically be a bundle of actual wires, as required by the parallel transfer of multi-bit data items. In this paper, however, we shall adopt serial data transfer.

The structure of the recursively defined boxes can be made explicit, and, if we conveniently assume $n \triangleq 2^m$, we obtain the network scheme of figure 2 (for $n = 8$). The reader will readily realize that such a network executes ALGORITHM in $m = \log_2 n$ steps, each consisting of a sequence (SHIFT;OPER), carried out respectively via the

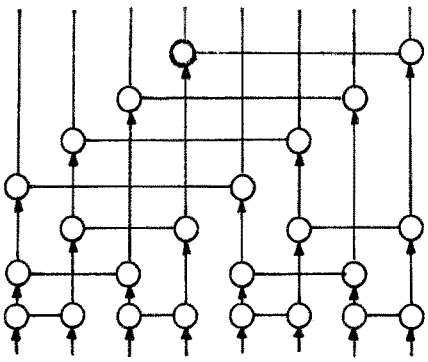

Figure 2. Explicit structure of the network.

vertical and horizontal wires. Each set of n/2 horizontal wires of identical length is called a sheaf, while the base-2 logarithm of the common length (a power of 2) is called the order of the sheaf. Note that at each step only one sheaf is active and all others are idle; thus the network naturally lends itself to pipelined operation.

To take full advantage of this pipeline capability, we must devise ways of solving a problem of size $N = ns$ on an n-input network. Of course, slicing the given ns-component vector into s n-component vectors, and feeding each of them through the network in pipeline fashion, corresponds to dividing the original problem into s subproblems and solving each of them separately. Note, however, that if $OPER_i$ is truly dependent upon i, when the n-component vectors corresponding to the s subproblems are pipelined, the OPER operation executed by a given module will have to be modified at each step. What remains to be seen is how the results of the s subproblems can be combined to yield the result of the original problem.

Before answering this question, we recall from [10] how a linear array — that is, an open-ended alignment of processing modules, with adjacent modules interconnected — can be used to implement ALGORITHM. If we obtain the iterative rendition of the latter, we observe that each data item in the array $T[0:2^r-1]$ is successively brought to interact with data items which are $2^0, 2^1, 2^2, \ldots, 2^{r-1}$ positions away. On the other hand, the postulated array system only has connections between adjacent positions. Thus the objective is to use the existing connections to rearrange the data in order to successively create the adjacencies required by the algorithm. The sequence of permutations realizing this objective is an appropriate sequence of "perfect shuffles" [13]. For example, a perfect-shuffle on an array of size 8 is achieved by the following sequence of "sets of exchanges":

```
initial     0 1 2 3↔4 5 6 7
            0 1 2↔4 3↔5 6 7
            0 1↔4 2↔5 3↔6 7
final       0 4 1 5 2 6 3 7
```

The realization of the perfect-shuffle on an array of size 2^j takes $2^{j-1}-1$ sets of exchanges. The reader will convince himself — or he can refer to [10] for details — that the desired sequence of adjacencies on an array of size 2^r are obtained by a sequence of perfect-shuffles on each of the 2^{r-j} contiguous subarrays of size 2^j, for j = 2,3,...,r. At the completion, however, the data items will be arranged according to the well known bit-reversal-permutation (BRP) [13], so that an identical sequence of perfect-shuffles restores the original arrangement. Thus we conclude that ALGORITHM can be executed on an array of length 2^r by means of a sequence of (SHUFFLE;OPER); it is also a simple exercise to realize that the total expended time is $O(2^r) \times$ time(OPER).

Reverting now to our original problem, suppose that the parameter s — the number of slices of the original data vector — is a power of 2, i.e., $s = 2^r$. We also make the convention to call forward a pipeline data movement proceeding from low order sheaves to high order ones (the reverse will obviously be called backward). Assuming forward pipeline, we now show that the combination of the results of the s subproblems can be done either by pre-processing the inputs to or by post-processing the outputs from the recursive network, i.e., by respectively placing an array of s modules either upstream or downstream of each of the network lines. The resulting networks are illustrated in figure 3. In either case the linear arrays will emulate the activity that in a larger network with ns input lines would be carried out by r sheaves. In the case of pre-processing (figure 3a), the arrays emulate the sheaves of order 0,1,2,...,r-1; in the case of post-processing (figure 3b), the emulated sheaves have order m, m+1, ..., m+r-1. Thus if we view the input data as represented by an s × n matrix of indices — running from 0 to (sn-1) — pre-processing and post-processing forward pipelines respectively correspond to the situations where the natural sequence 0,1,...,(sn-1) is obtained by reading the index-matrix in column-major and row-major order. We leave it as an exercise for the reader to devise the index matrix of arrangements corresponding to executing ALGORITHM on the networks of figures 3 by feeding the data backward. Having observed that there are four equivalent ways of correctly obtaining a pipeline implementation of ALGORITHM (pre- and post-processing, forward and backward pipeline), from now on we shall refer to the preprocessing/forward feeding network as standard.

In a standard network, the pre-processing phase is completed in O(s) steps while the pipelining phase requires max(s,m) additional steps. Therefore, if $s \geq m$ the running time is O(s) steps. It is worth mentioning at this point that the cube-connected-cycles architecture described in [10] is basically analogous to the

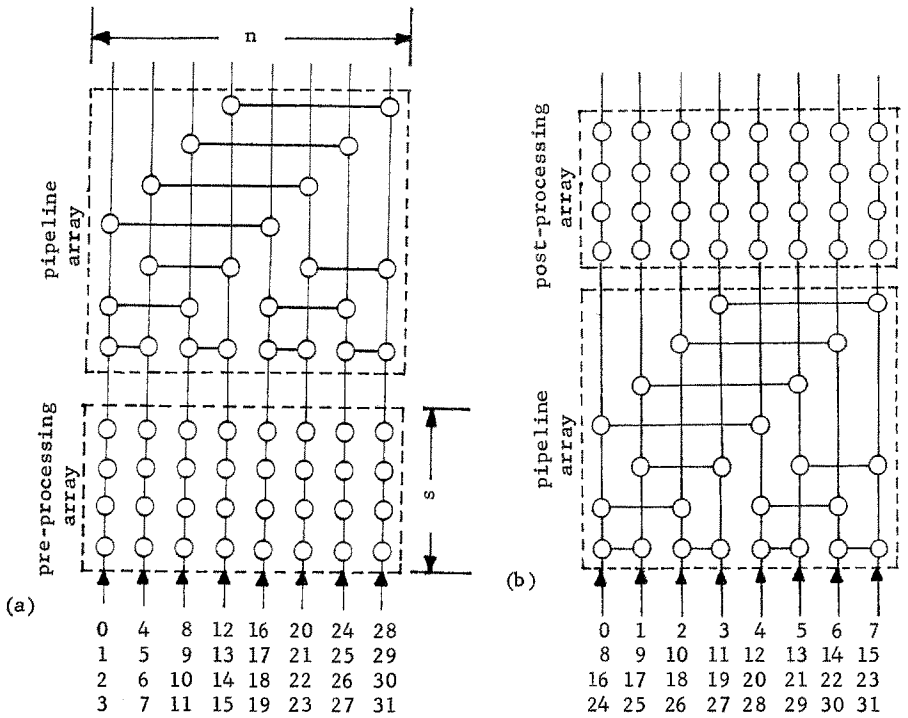

Figure 3. Structures of forward pipeline networks, with pre-(a) and post-processing (b). Also shown are the feeding arrangements of the data ($n = 8$, $s = 4$).

network just described, with the added twist that the function of pre-processing (or post-processing) and pipeline processing are shared by the same set of modules, arranged as cycles rather than as open-ended alignments (so that pre-processing and pipelining can be correctly sequenced).

Finally we consider the area of layouts of the networks in figure 3. The layouts comply with the requirement of the commonly accepted VLSI model of computation [3,4]. Specifically, wires are laid out on a planar grid and each wire had width λ, the grid spacing. Modules are assumed to have fixed horizontal size W (width) but vertical size H (height) which may be arbitrary. We consider the standard placement shown in figure 4. The total width is clearly nW. As to the height, in the pre-processing arrays the modules may be laid out in a very compact fashion thereby obtaining a total height sH; in the pipeline array the sheaf of order i contributes a height $H + \lambda \cdot 2^i$ (H for the modules, $\lambda \cdot 2^i$ for the wires). Thus the height contributed by the m-sheaves become $Hm + (2^m-1)\lambda$, and the total chip height is $(s+m)H + (2^m-1)\lambda < (s+m)H + 2^m \cdot \lambda$. Note that, as long as $H \leq \frac{2^m}{s+m}\lambda$, the total height of the layout remains $O(2^m)$ and is unaffected by the module height.

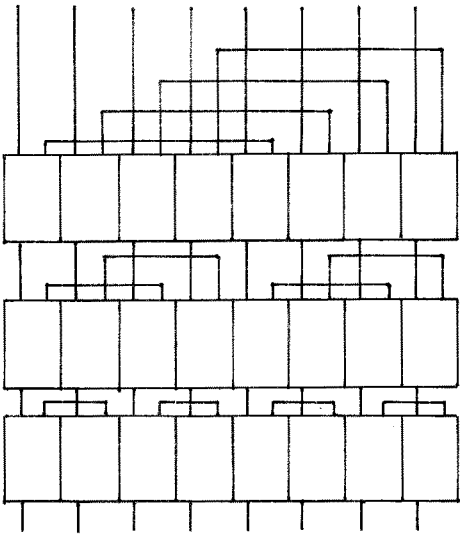

Figure 4. Placement of modules in the network.

For ease of reference, the just described type of network, i.e., pre-processing (or post-processing) array and pipeline array, will be called the cube-connected-arrays (CCA), by analogy with the structure described in [10]. Width of a CCA is the number of input lines, while its depth is the number of modules in each of the pre- or post-processing arrays.

3. Circuits for the DFT

We can now specialize the general network scheme presented in Section 2 to the computation of the Discrete-Fourier-Transform (DFT) of a vector $\underline{a} = [a_0, a_1, \ldots, a_{N-1}]$, of integers, where $N = ns$, $n = 2^m$, $s = 2^r$. Given a suitably chosen field F, endowed with a primitive root of unity, ω, of order N, $\underline{A} = [A_0, A_1, \ldots, A_{N-1}]$ denotes the DFT of $[a_0, a_1, \ldots, a_{N-1}]$ over F. All operands (i.e., inputs, outputs and intermediate results) are supposed to be represented with O(logN) bits. After subjecting the components of $\underline{a}$ to the bit-reversal-permutation, and storing the permuted $\underline{a}$ in T[0:N-1], $\underline{A}$ is computed by the following FFT algorithm:

```
proc    FFT(0,N-1)
begin   FFT(0,N/2-1), FFT(N/2,N-1);
        foreach i: 0 ≤ i < N/2 par do  (T[i],T[i+N/2]) ← (T[i]+ω^i T[i+N/2],
                                                          T[i]-ω^i T[i+N/2])
end
```

This procedure conforms exactly with our general paradigm of ALGORITHM, so that it

can be executed by a CCA-network of width 2^m and depth $(2^r + m)$. Special care, however must be exercised in programming OPER for any given sheaf of the pipeline array.

For ease of reference, we index the modules $\{M_j : 0 \le j < mn\}$ of the pipeline array in row-major order from 0 to mn-1, each sheaf representing a row (figure 5). By contrast, the BRP-permuted $\underline{a}$ is fed in column-major order (see figure 5).

Let $\varphi = \omega^{n/2}$. Then, the powers of ω used by module $M_{(m-1)n+j}$, $(0 \le j < n/2)$, in the pipeline operation, will successively be

$$\omega^j, \omega^j\varphi, \ldots, \omega^j\varphi^{s-1}$$

while $M_{(m-1)n+j}$ $(n/2 \le j < n)$ will have the passive role previously described. It is also relatively easy to realize that, in general, module M_{in+j} $(0 \le i < m;$ $0 \le j < 2^i)$ will successively use powers $\omega^{j\cdot 2^{m-1-i}}$, $\omega^{j\cdot 2^{m-1-i}} \cdot \varphi, \ldots, \omega^{j\cdot 2^{m-1-i}} \cdot \varphi^{s-1}$ and that M_{in+j} $(0 \le i < m;\ 2^i \le j < 2^{i+1})$ has a passive role. Moreover, module M_{in+j} $(0 \le j < n)$ behaves exactly as module $M_{in+j \bmod 2^{i+1}}$. Thus, in each active module, say M_{in+j}, we only need store <u>two</u> constants: $\omega^{j\cdot 2^{m-1-i}}$ and φ. During each step of the pipeline operation, the combining coefficient α is updated through the

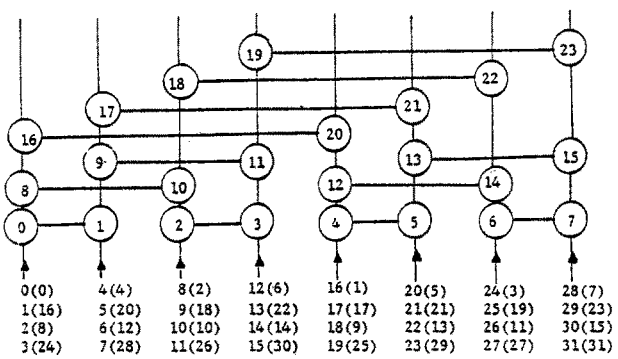

Figure 5. Labeling of modules and data. Data are fed in column-major order, after a BRP (shown within parentheses).

single multiplication $\alpha \leftarrow \alpha\cdot\varphi$. We conclude that each active module performs at each step the following operations on two data items (U,V):

1. $V \leftarrow \alpha\cdot V$
2. $(U',V') \leftarrow (U+H, U-V)$
3. $\alpha \leftarrow \alpha\varphi$.

Therefore each module must contain four operand registers and a classical shift-and-add multiplier. The latter is basically a linear systolic array (see, e.g., [7]) where the two operands proceed in opposite directions; the process obtains the convolution of the two operands viewed as binary polynomials and transforms the

convolution into the ordinary product. The multiplier contains a number of cells proportional to the product length, and can therefore be laid out in a rectangle of constant width $O(1)\cdot\lambda$ and height proportional to the operand length. The same holds for the registers. Thus, each module can be laid-out in a rectangle $O(1) \times O(\log N)\cdot\lambda^2$, since $O(\log N)$ is the assumed operand length.

If we now choose s to be a power of 2 in the interval $[m, 2^m/m]$, it follows that the network width is $O(2^m)$, while the network height — according to the standard placement of figure 4 — is bounded above by $(s+m)O(\log N) + 2^m\cdot\lambda$. Since $N = ns$, we obtain that, for all choices of s, the height is also $O(2^m)$, whence the chip area is $O(2^{2m})$.

As to the computation time, we have O(s) steps both in the pre-processing array and in the pipeline array, each of which uses $O(\log N)$ time (for multiplication and serial operand transfer). Thus computation is completed in time $O(s\log N)$.

Finally, we consider the performance measure AT^2. We have (note that $2^m = N/s$):

$$AT^2 = O(2^{2m})\cdot[O(s\log N)]^2 = O(\frac{N^2}{s^2})O(s^2\log^2 N) = O(N^2\log^2 N)$$

thereby achieving the lower-bound of Thompson [2]. Thus, the class of FFT-circuits presented is optimal with respect to the AT^2 measure of complexity. Note that as s varies in the interval $[m, 2^m/m]$ the computation time T varies in an interval $[O(\log^2 N), O(\sqrt{N/\log N})]$.

Note finally that the network deployed in figure 5 is of the standard type (pre-processing/forward feeding). An alternative implementation, deploying a pre-processing/backward feeding network, receives the input sequence in natural order and produces a BRP-permuted transform.

4. Circuits for integer multiplication

We now apply the results obtained on the DFT in the preceding section to integer multiplication. It is well known that the DFT allows one to compute convolution products, and this technique has already been used to develop methods for integer multiplication [14]. Indeed, Schönhage and Strassen have demonstrated that, given two sequences of $q/2$ integers in the range $[0,q]$, if $\omega = e^{2\pi i/q}$ and its powers are represented with $5\log_2 q$ bits, and the arithmetic is carried out with this precision, the approximation error in computing convolutions via the FFT remains confined to the fractional part of the results. An alternative to this scheme, with a different choice of the field over which the DFT is computed, has been recently outlined by Brent and Kung [4]. Their scheme consists in carrying out the arithmetic in the prime field F_m, where the prime m has the form $m = pq + 1$ and $\log p = O(\log q)$; F_m clearly contains a primitive root of unity of order q. In both schemes one deals with operands represented with $k\log q$ bits, for some small constant k.

In order to compute the product of two N-bit nonnegative integers, we divide each operand into $\frac{q}{2} = \frac{N}{\log N}$ blocks of $\log N \simeq \log q$ bits each and compute the q-term cyclic convolution of the two sequences via the FFT. In order to examine the details, we review this well-known process (here a and b are the two q-term sequences):

1. Compute A, the q-term FFT of a.
2. Compute B, the q-term FFT of b.
3. Compute C, the term-by-term product of A and B.
4. Compute c, the FFT^{-1} of C. The sequence c, the cyclic convolution of a and b, has (q-1) terms (the rightmost term is always equal to 0), each represented with 3logq bits.
5. Release the carries.

The last step – "Release the carries" – denotes the usual transformation of a convolution product into an integer product: indeed, each of the terms of the convolution sequence is 3logq bits long, while two consecutive terms have weights only logq bit positions apart, as shown in figure 6(a). By arranging the terms of the

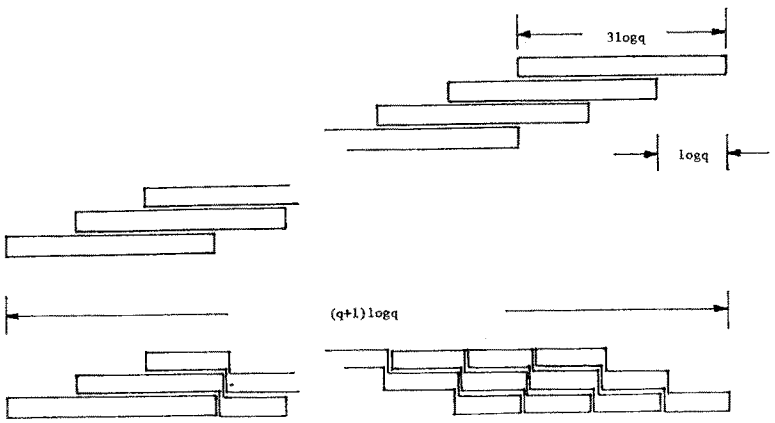

Figure 6. Illustration of the carry-release operation.

convolution product as shown in figure 6(b), it is obvious that releasing the carries corresponds to the addition of three (q+1)logq-bit binary numbers. The necessity to implement with ease this operation dictates the feeding arrangement of the operands, as we shall now examine.

To execute the operation of figure 6(b), we propose to deploy a modified Brent-Kung binary adder [4], the modification consisting in that the elementary adder is now an (area O(logq), time O(logq)) logq-bit adder, rather than an ordinary full adder. Referring to [4], the global adder has area $A = O((q + \frac{q}{s}\log\frac{q}{s})\log q)$ and time $T = O((s + \log\frac{q}{s})\log q)$, where s has the usual meaning and may vary in the

interval $[\log q, \sqrt{\frac{q}{\log q}}]$. It follows that, for this range of s, the maximum value of AT^2 is $O(q^2 \log^2 q)$, if we consider the adder alone.

The terms of the convolution sequence must be fed to this adder — in pipeline fashion — in their natural order, column-major. This implies that in the same order we must feed both of the original operands, whence the FFT network must be of the pre-processing/backward pipeline type; by contrast the FFT^{-1}-network will be of the pre-processing/forward pipeline type. The entire arrangement is shown in figure 7, where the pre-processing stages of the FFT^{-1}-network are also used to compute the term-by-term product of the transforms.

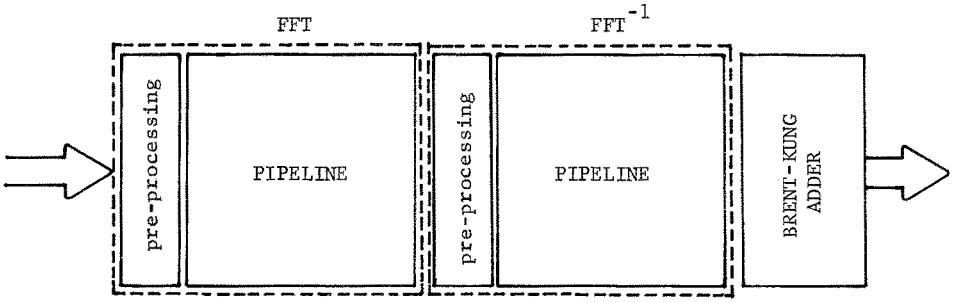

Figure 7. Structure of the integer multiplier.

To evaluate the performance of the proposed arrangement, we recall from Section 3 that the DFT of a sequence of q terms, each represented with logq bits, can be computed by a circuit with the following characteristics:

$$AT^2 = O(q^2 \log^2 q) \text{ for any T such that } O(\log^2 q) \leq T \leq O(\sqrt{q \log q}).$$

For all selectable values of T, the contribution of the carry-release circuit to AT^2 is at worst of the same order. Therefore, recalling our original choice $q/2 = N/\log N$, we have designed circuits for N-bit binary integer multiplication having the characteristics:

$$AT^2 = O(N^2) \text{ for any T such that } O(\log^2 N) \leq T < O(\sqrt{N}).$$

These circuits meet the $AT^2 = \Omega(N^2)$ bound of Brent and Kung [4]; in this class, the slow circuit corresponding to $T = O(\sqrt{N})$ also meets the $AT = \Omega(N^{3/2})$ lower bound.

Although the proposed circuits appear to be too complex for being feasible on one chip in the present state of technology, the sheer existence of, say, an $A = O(N)$, $T = O(\sqrt{N})$ multiplier raises interesting very practical prospects.

References

[1] C. Mead and L. Conway, Introduction to VLSI Systems, Addison-Wesley, Reading, Mass., 1979.

[2] C. D. Thompson, "Area-time complexity for VLSI," Proc. of the 11th Annual ACM Symposium on the Theory of Computing (SIGACT), pp. 81-88, May 1979.

[3] C. D. Thompson, "A complexity theory for VLSI," Ph.D. Thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Penn., 1980.

[4] R. P. Brent and H. T. Kung, "The chip complexity of binary arithmetic," Proc. of 12th ACM Symposium on Theory of Computing, (Los Angeles), pp. 190-200, May 1980.

[5] J. E. Vuillemin, "A combinatorial limit to the computing power of VLSI circuits," Proc. 21st Symposium on Foundation of Computer Science, (Syracuse), pp. 294-300, Oct., 1980.

[6] H. Abelson and P. Andreae, "Information transfer and area-time trade-offs for VLSI multiplication," Communications of the ACM, vol. 23, n. 1, pp. 20-22; January, 1980.

[7] H. T. Kung and C. E. Leiserson, "Algorithms for VLSI processor arrays," Symposium on Sparse Matrix Computations, Knoxville, Tenn., Nov. 1978.

[8] L. J. Guibas, H. T. Kung, and C. D. Thompson, "Direct VLSI implementation of combinatorial algorithms," Proc. Conference on VLSI Architecture, Design, Fabrication, Calif. Inst. of Techn., January 1979.

[9] J. E. Savage, "Area-time tradeoffs for matrix multiplication and transitive closure in the VLSI model," Proc. of the 17th Annual Allerton Conference on Communications, Control, and Computing, October 1979.

[10] F. P. Preparata and J. Vuillemin, "The cube-connected-cycles: a versatile network for parallel computation," (Extended Abstract), Proceedings of 20-th Annual IEEE Symposium on Foundations of Computer Science, Puerto Rico, October 1979.

[11] F. P. Preparata and J. Vuillemin, "Area-time optimal VLSI networks for multiplying matrices," Information Processing Letters, vol. 11, n. 2, pp. 77-80, October 1980.

[12] F. P. Preparata and J. Vuillemin, "Optimal integrated-circuit implementation of triangular matrix inversion," 1980 International Conference on Parallel Processing, Boyne, Mich., pp. 211-216, Aug. 1980.

[13] H. S. Stone, "Parallel processing with the perfect shuffle," IEEE Transactions on Computers, vol. C-20, pp. 153-161; 1971.

[14] A. Schönhage and V. Strassen, "Schnelle Multiplikation grosser Zahlen," Computing, vol. 7, pp. 281-292; 1971.