Reprinted from INFORMATION AND CONTROL
Vol. 58, Nos. 1–3, July/August/September 1983

Printed in Belgium

Area–Time Optimal VLSI Integer Multiplier with Minimum Computation Time*

KURT MEHLHORN

Universität der Saarlandes, Fachbereich 10, Saarbrücken, West Germany

AND

FRANCO P. PREPARATA

Coordinated Science Laboratory, University of Illinois, Urbana, Illinois 61801

According to VLSI theory, $[\log n, \sqrt{n}]$ is the range of computation times for which there may exist an AT^2-optimal multiplier of n-bit integers. Such networks were previously known for the time range $[\Omega(\log^2 n), O(\sqrt{n})]$; this theoretical question is settled, by exhibition of a class of AT^2-optimal multipliers with computation times $[\Omega(\log n), O(\sqrt{n})]$. The designs are based on the DFT on a Fermat ring, whose elements are represented in a redundant radix-4 form to ensure $O(1)$ addition time.

1. INTRODUCTION

Research on efficient integer multiplication schemes, potentially suitable for direct circuit implementation, has been going on for some years. Investigations have focussed on both the realization of practical (and possibly suboptimal) networks and the more subtle question of the existence of optimal networks. Optimality is defined with respect to the customary AT^2 measure of complexity, which is central to the synchronous VLSI model of computation (Thompson, 1979; Brent and Kung, 1981). Here A is the area of the multiplier chip, while T is the computation time, i.e., the time elapsing between the arrival of the first input bit and the delivery of the last output bit. As is well known (Abelson and Andreae, 1980; Brent and Kung, 1981), any multiplier of two n-bit integers must satisfy $AT^2 = \Omega(n^2)$ and $A = \Omega(n)$ in the VlSI model; in addition, standard fan-in constraints of the logic gates yield the lower bound $T = \Omega(\log n)$. These three lower bounds

* This work was supported by the National Science Foundation under Grants MCS-81-05552 and ECS-81-06939; additional support was provided by the Deutsche Forschungsgemeinschaft SFB 124, VLSI-Entwurf und Parallelität.

0019-9958/83 $3.00

indicate that $[\log n, \sqrt{n}]$ is the range of computation times for which there may exist an AT^2-optimal multiplier.

The search for an AT^2-optimal integer multiplier began with the suboptimal design of Brent and Kung (1981), for which $AT^2 = O(n^2 \log^3 n)$. Subsequently, Preparata and Vuillemin (1981b) proposed a class of optimal designs whose computation time could be selected in the range $[\theta(\log^2 n), \theta(\sqrt{n})]$. More recently, Preparata (1983) exhibited an optimal mesh-connected multiplier achieving $T = O(\sqrt{n})$. An intriguing feature of all the above optimal designs is the explicit recourse to the Discrete Fourier Transform (DFT), as the device used for computing convolutions. However, none of these optimal designs achieves the minimum computation time $T = \theta(\log n)$. On the other hand there are well-known multiplication algorithms which achieve optimum computation time $T = \theta(\log n)$, e.g., the Wallace tree (1964) and Dadda counting (1965). Both algorithms are not easily embedded into silicon because of their irregular interconnection pattern. More recently, there have been proposals of desings with optimum computation time and nearly optimum AT^2-measure (Vuillemin, 1983; Becker, 1982; Luk and Vuillemin, 1983; Lengauer and Mehlhorn, 1983). Moreover, some of these designs are eminently practical. We refer the reader to Luk and Vuillemin (1983) for a detailed discussion. All of these designs are based on divide-and-conquer techniques and achieve their speed by the use of a redundant operand representation, which results in $O(1)$ addition time. The most efficient of these designs (Luk and Vuillemin, 1983; Lengauer and Mehlhorn, 1983) achieves $T = O(\log n)$ and $AT^2 = O(n^2(\log n)^2)$.

In this paper we shall exhibit a class of optimal, i.e., $AT^2 = \theta(n^2)$, designs realizing any computation time in the range $[\Omega(\log n), O(\sqrt{n})]$, thereby realizing the first AT^2-optimal $O(\log n)$-time multiplier. More generally, the new design settles, at least theoretically, the problem of integer multiplication: there exist optimal designs for the entire spectrum of possible computation times. Our new design incorporates ideas of many of the papers cited above. Not unlike previous optimal designs, it makes essential use of the DFT over a finite ring G, which we choose as a Fermat ring. In contrast to previous papers, a low-order DFT is used to achieve fast computation time. More precisely, in order to achieve computation time $O(T)$ we will resort to a T-point DFT over a ring of $2^{O(n/T)}$ elements. In Preparata and Vuillemin (1981b) an $(n/\log n)$-point DFT over a ring of $2^{O(\log n)}$ elements is used for all achievable values of T. Since we compute a DFT in a large ring of $2^{O(n/T)}$ elements, efficient implementations of the ring operations and of the data transfer between computing elements are crucial. We borrow from Preparata (1983) the idea of computing the DFT on a mesh of processing elements. Only communication between adjacent processing elements is required in this case and hence we can provide for a large communication bandwidth without paying too high a penalty in area. Each processing

element of the mesh has the ability to do additions–subtractions over G and multiplications by powers of the root of unity. We choose a redundant representation for ring elements and thus achieve $O(1)$ addition/subtraction time in a small area. Since ring G is a Fermat ring, i.e., the set of integers modulo $m = 2^p + 1$ for some p, and since the root of unity used in the DFT is a power of two, multiplication by a power of the root of unity can be essentially reduced to a cyclic shift and a small number of additions/subtractions. We implement cyclic shifts by means of a cube-connected-cycle network (Preparata and Vuillemin, 1981a). Finally, general multiplications in G are realized by one of the fast, suboptimal designs referred to above. Since only $O(T)$ general multiplicaitions of ring elements (which are essentially $O(N/T)$-bit numbers) are required, we will stay within the desired limits of time and area.

The paper is organized as follows. In Section 2 we review the arithmetic basis for the proposed multiplication scheme. We start with a description of one of the fast designs based on divide-and-conquer. Next we review how integer multiplication can be reduced to polynomial multiplication (convolution). We will then discuss the school-method (for polynomial multiplication) and derive from it a $T = O(\log n)$, $A = O(n^2/\log n)$ design. This design is probably the most practical design proposed in this paper. Finally, we discuss interpolation/evaluation schemes and more specifically the DFT for computing convolutions efficiently. In Section 3 we describe the proposed multiplication scheme in detail. We first review how to compute the DFT on a mesh and then discuss in detail the organization of each mesh module. Appendix 1 contains a discussion of a redundant number representation which we use for the algorithm and Appendix 2 shows how to compute the DFT on a mesh; the latter material is essentially taken from Preparata (1983). Appendix 3 contains a succinct review of the structure and operation of the cube-connected-cycles network.

2. Arithmetic Background

In this section we briefly review the arithmetic basis of the proposed multiplication scheme. Specifically, we recall how integer multiplication can be solved by divide-and-conquer techniques and more generally by polynomial multiplication. We will also review a particular VLSI-design based on divide-and-conquer techniques. We will then show how the "school-method" for polynomial multiplication can be used to construct a $T = O(\log n)$, $AT^2 = O(n^2 \log n)$ multiplier. Finally, we will discuss how convolution can be computed by an evaluation/interpolation scheme and we shall describe one particularly efficient specialization of the latter as a Discrete Fourier Transform over a Fermat ring.

Throughout this paper a and b are nonnegative integers in the range $[0, 2^{n/2}-1]$ (n even). We use $c = a \times b$ to denote their product and $a_{n-1},\ldots,a_0$, $b_{n-1},\ldots,b_0$, $c_{n-1},\ldots,c_0$, where $a_{n/2} = \cdots = a_{n-1} = b_{n/2} = \cdots = b_{n-1} = 0$, to denote the binary representations of a, b, and c, respectively.

2.1. *Integer Multiplication by Divide-and-Conquer*

Assume that n is divisible by 4 for the sake of simplicity. We can then write

$$a = a_1 2^{n/4} + a_0, \qquad b = b_1 2^{n/4} + b_0, \qquad c = c_2 2^{n/2} + c_1 2^{n/4} + c_0$$

with $a_1, a_0, b_1, b_0 \in [0, 2^{n/4}-1]$ and $c_2 = a_1 b_1$, $c_1 = a_0 b_1 + a_1 b_0$, $c_0 = a_0 b_0$. Hence we can compute the product of two $n/2$-bit numbers by computing four products of $n/4$-orbit numbers and a few sums of n-bit numbers. The next observation to make is that addition takes time $O(1)$ if a suitable redundant representation is used (cf. Appendix 1). Hence this scheme will result in a $T = O(\log n)$ multiplier. A VLSI layout with $A = O(n^2 \log n)$ is readily obtained and can be found in Luk and Vuillemin (1983).

An interesting improvement upon the technique described above is due to Karazuba and Ofman (1962). They observed that c_2, c_1, and c_0 can be expressed as

$$c_2 = a_1 b_1 \qquad c_1 = (a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0 \qquad c_0 = a_0 b_0$$

and hence *three* multiplications of numbers of half the length suffice. Again, if combined with a redundant number representation, a $T = O(\log n)$ multiplier results. Also, a VLSI layout with $A = O(n^2)$ is available and can be found in Luk and Vuillemin (1983), and Lengauer and Mehlhorn (1983). Thus $AT^2 = O(n^2 \log^2 n)$.

It is important to note that both of the above multipliers are pipelinable (in technical terms, their periods are $O(1)$), since at each step of the computation all data lie on a *single* level of the recursion. Thus the *pipelined 3-multiplication multiplier* (P3M) can be used to multiply $O(\log n)$ pairs of n-bit numbers in time $O(\log n)$. We will exploit this observation below.

2.2. *Integer Multiplication via Polynomial Multiplication*

Integer multiplication by divide-and-conquer is a special case of integer multiplication via polynomial multiplication. Let k be an integer, $k \geqslant 2$. For the divide-and-conquer scheme we have $k = 2$. Assume w.l.o.g that k divides n. We subdivide the binary representation $a_{n-1} \cdots a_0$ of a into k strings of length n/k each and consider each string as the representation of a binary number in the range $[0, 2^{n/k}-1]$. In this way we associate with integer a the

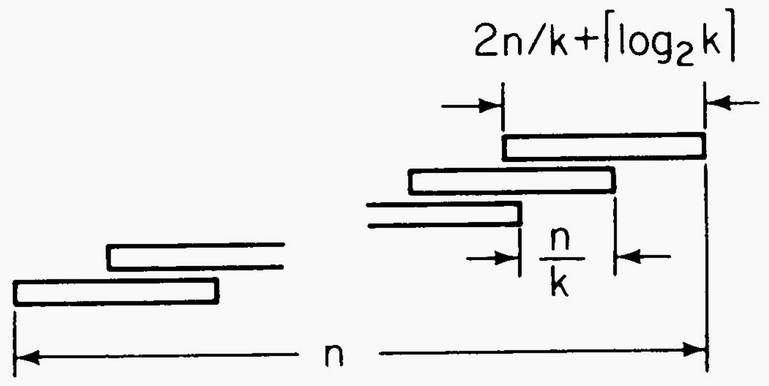


FIG. 1. Illustration of the release-of-the-carries.

polynomial $A(x) = \sum_{j=0}^{k-1} A_j x^j$ such that $A_j \in [0, 2^{n/k} - 1]$, $a = A(2^{n/k})$ and $A_{k/2} = \cdots = A_{k-1} = 0$. Similarly we associate polynomial $B(x) = \sum_{j=0}^{k-1} B_j x^j$ with b. Note that $A(x)$ and $B(x)$ are really of degree $\lceil k/2 \rceil - 1$. Let $C(x) = A(x) \cdot B(x)$ be their product. Then $C(x)$ is of degree $k-1$ and $C(2^{n/k}) = A(2^{n/k})\, B(2^{n/k}) = ab$. Also, each coefficient of $C(x)$ lies in the range $[0, k2^{2n/k} - 1]$ and can thus be expressed as a $(2n/k + \lceil \log_2 k \rceil)$-bit number. It follows that the product ab can be obtained by expressing each coefficient of $C(x)$ as a $(2n/k + \lceil \log_2 k \rceil)$-bit number, by positioning the coefficients n/k bits apart as shown in Fig. 1 and by adding these $(k-1)$ numbers. This transformation of $C(x)$ into $c = ab$ is normally referred to as the "release-of-the-carries" and can be performed in time $O(\log n)$ (Preparata and Vuillemin, 1981b).

At this point we have reduced integer multiplication to polynomial multiplication. The naive method for the latter problem is the school-method: compute the k^2 products $A_i B_j$, $0 \leqslant i, j < k$, and sum appropriate terms. For $k = 2$ this leads to the 4-multiplication recursive scheme described at the beginning of Section 2.1.

We will next show how to combine the P3M multiplier with the "school-method" for convolution in order to obtain an $T = O(\log n)$, $AT^2 = O(n^2 \log n)$ VLSI-multiplier. We will describe two quite different methods.

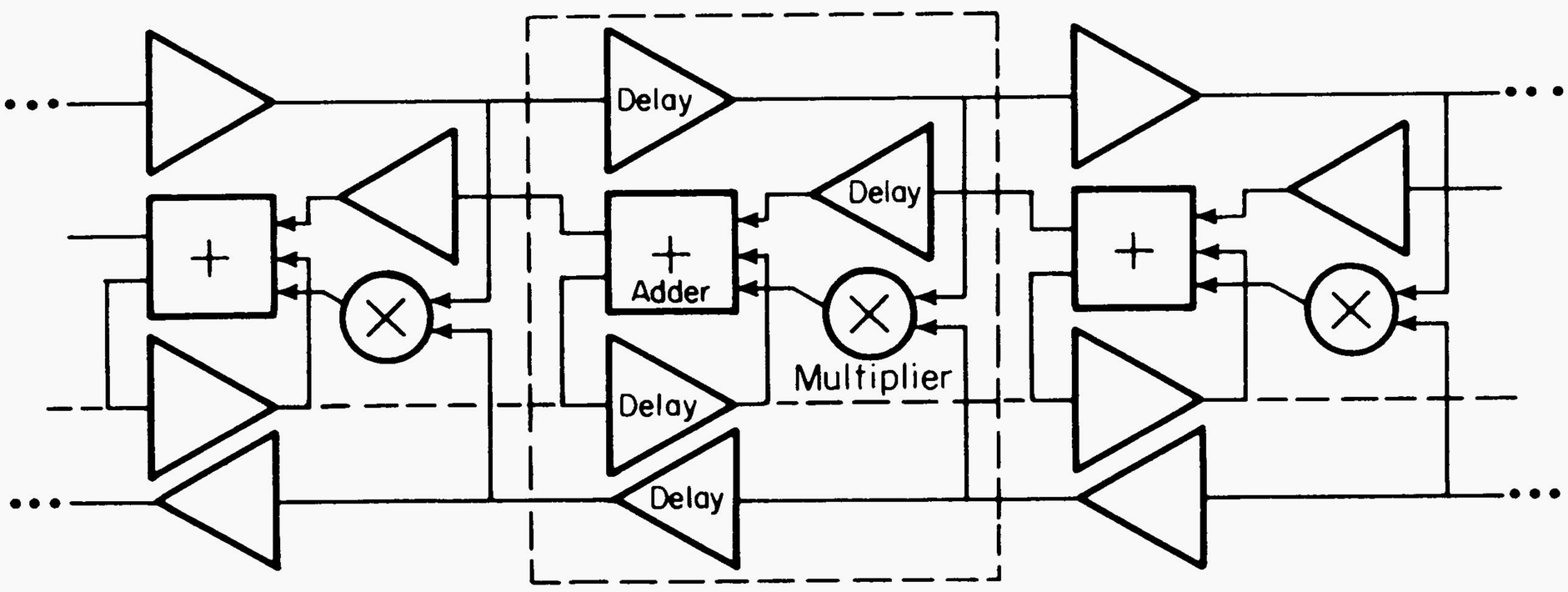


FIG. 2. Structure of Muller's Multiplier.

The first method is a hybridization with a multiplier due to Muller (1963). The multiplier originally proposed by Muller computes the product of two s-digit integers by convolving the two factors (see Fig. 2). It consists of $2s-1$ cells, and the product is obtained in $2s-1$ shifts and adds. In the binary case, the adder is a conventional full binary adder and the "elementary multiplier" is just an AND-gate. Suppose now we subdivide each of the n-bit operands into $\lceil \log_2 n \rceil$ strings of $(n/\lceil \log_2 n \rceil)$-bits each to be viewed as a binary number. We now construct a Muller multiplier with $2\lceil \log_2 n \rceil - 1$ cells, each of which is adapted to process $(n/\lceil \log_2 n \rceil)$-bit numbers, rather than single bits. The adaptation consists of replacing the AND-gate by a P3M multiplier for $n/\lceil \log_2 n \rceil$-bit numbers, and the full adder by a three-operand adder; the redundant representation is kept throughout, so that $O(1)$-time addition is guaranteed. The sequential feeding of the $\lceil \log_2 n \rceil$ strings (each fed in parallel) provides the pipeline input to the P3M multipliers, so that the overall multiplication is completed in time $O(\log n)$. As to the chip area, each of the $2\lceil \log_2 n \rceil - 1$ cells has area $O(n^2/\log^2 n)$, thereby resulting in an overall area $O(n^2/\log n)$. It follows that $AT^2 = O(n^2 \log n)$, as claimed earlier.

Remark. Of course, the P3M multiplier used in the design above can be replaced by any other pipelinable fast multiplier. In particular, we might use the $T = O(\log n)$, $A = O(n^2 \log n)$ multiplier described in Vuillemin (1983), Luk and Vuillemin (1983), and Becker (1982), and obtain a $T = O(\log n)$, $A = O(n^2)$ multiplier. This design is probably the most practical design proposed in this paper.

An alternative approach has been described in Lengauer and Mehlhorn (1983) and is as follows. Divide the n-bit integers a and b into $k = (\log_2 n)^{1/2}$ strings of $n/(\log_2 n)^{1/2}$ bits each. We now use a P3M multiplier for $n/(\log_2 n)^{1/2}$-bit integers in order to compute the $k^2 = \log_2 n$ products $A_i B_j$. Adding up appropriate terms and releasing the carry finishes the multiplication. It is easily seen that the area of the design is dominated by the area of the P3M multiplier and hence is $O(n^2/\log n)$. Thus a $T = O(\log n)$, $A = O(n^2/\log n)$ multiplier results.

We have now described two alternative implementations of the school-method. The first one uses $k = \log_2 n$ and the second one uses $k = (\log_2 n)^{1/2}$. In the first implementation we use $\log n$ P3M multipliers to compute the $(\log_2 n)^2$ multiplications of $(n/\log_2 n)$-bit integers and in the second implementation we use one P3M multiplier to compute the $\log_2 n$ multiplications of $n/(\log_2 n)^{1/2}$-bit integers.

At this point it is natural to hope that even more efficient multipliers can be obtained by replacing the school-method for polynomial multiplication by a more efficient scheme. The more efficient schemes are based on the concept of evaluation–interpolation and are described in the next section.

2.3. *Polynomial Multiplication via Evaluation/Interpolation and the Discrete Fourier Transform*

Let G be a division[1] ring and let $A(x)$, $B(x)$ be polynomials of degree $<k/2$ over G. Let $x_0, x_1, \ldots, x_{k-1}$ be distinct elements of G. Denoting, as usual, by $C(x)$ the degree-$(k-1)$ product $A(x) \cdot B(x)$, we have

$$C(x_j) = A(x_j) \cdot B(x_j) \qquad 0 \leqslant j \leqslant k-1.$$

Thus, by *evaluating* $A(x)$ and $B(x)$ at each of the values $x_0, \ldots, x_{k-1}$, we obtain, by means of *only* k multiplications over G, the values $C(x_0), \ldots, C(x_{k-1})$, from which the k coefficients of C are computed by *interpolation*.

Karazuba's 3-multiplication method is an instance of evaluation/interpolation for $k=3$. Let $A(x) = A_1 x + A_0$, $B(x) = B_1 x + B_0$ be two polynomials of degree 1. We choose $x_0 = 0$, $x_1 = 1$ and $x_2 = \infty$. Then $A(x_0) = A_0$, $A(x_1) = A_1 + A_0$, $A(x_2) = A_1$, $B(x_0) = B_0$, $B(x_1) = B_1 + B_0$, $B(x_2) = B_1$, and $C(x_0) = A_0 B_0$, $C(x_1) = (A_1 + A_0)(B_1 + B_0)$, $C(x_2) = A_1 B_1$. Finally, the interpolation formulae are

$$C_0 = C(x_0)$$

$$C_1 = C(x_1) - C(x_0) - C(x_2)$$

$$C_2 = C(x_2).$$

As one can see from Karazuba's metod, the choice of points $x_0, \ldots, x_{k-1}$ is very crucial for the effectiveness of the evaluation/interpolation scheme. It is well known that for large k a good choice for the evaluation points is consecutive powers of an order $-k$ primitive root of unity in G. This leads to the Discrete Fourier Transform (DFT) (Aho, Hopcroft, and Ullman, 1974).

In particular we want to choose the commutative ring G such that if ω is a primitive root of unity of order k, multiplication of an element of G by ω^i $(i = 0, \ldots, k-1)$ can be done very efficiently. One very attrative choice is provided by a Fermat ring, i.e., by the set of the integers modulo a number of the form $2^p + 1$, for integer p: indeed, as we show in Appendix 1, multiplication by ω^i reduces to a minor variant of left cyclic shift. The suitability of a Fermat ring to our objective is demonstrated by the following property (see Aho *et al.*, 1974, p. 266, Theorem 7.5):

PROPOSITION 1. *Let r and ω be powers of 2 and let $m = \omega^{r/2} + 1$. Letting $\mathbf{Z}_m$ be the ring of integers modulo m (a Fermat ring), then r and ω*

[1] Below, this condition on the nature of G will be relaxed.

have multiplicative inverses in $\mathbf{Z}_m$ *and* ω *is a primitive* r*th root of unit in* $\mathbf{Z}_m$.

The arithmetic of Fermat rings is described in Appendix 1.

We close this section with a brief description of the construction we are about to describe. Let T be an integer between $\log n$ and $\sqrt{n}$ (the symbol T is chosen as a reminder that this integer is the "target computation time" of the network). We reduce multiplication of n-bit integers to multiplication of polynomials of degree T with coefficients in the range $[0, 2^{n/T})$. Multiplication of polynomials is based on evaluation/interpolation over a Fermat ring. For the T multiplications in the ring we use one P3M multiplier for (n/T)-bit integers. Thus all T multiplications take time $O(T + \log n/T) = O(T)$ and area $A = O(n^2/T^2)$. Evaluation and interpolation are the DFT and its inverse. Section 3 is devoted to the computation of the order $-T$ (DFT) in a ring of size $2^{O(n/T)}$ in time $O(T)$ and area $O(n^2/T^2)$.

3. The Multiplier Network

A multiplier network of the type we describe below consists of four major subnetworks, illustrated in Fig. 3. Operands are loaded from the left and intermediate results migrate to the right, residing a certain amount of time in each of the four major subnetworks. Operands are represented with n bits and, denoting by $T \in [\log n, \sqrt{n})]^2$ a *power of* 2 *that divides* n, each operand is subdivided into T strings of n/T bits. For two such operands a and b we have

$$a = \sum_{i=0}^{T-1} a_i 2^{ni/T}, \qquad b = \sum_{i=0}^{T-1} b_i 2^{ni/T},$$

where $0 \leqslant a_i,\ b_i < 2^{n/T}$ for $i = 0,\ldots, T/2 - 1$ and $a_i = b_i = 0$ for $i \geqslant T/2$. Analogously we define $c = a \times b$, so that

$$c_j = \sum_{i=0}^{j} a_i b_{j-i} \qquad 0 \leqslant j < T.$$

From this expression, it is obvious that $0 \leqslant c_j < T2^{2n/T}$. We now wish to treat the a_i's, b_i's, and c_i's as members of a Fermat ring $\mathbf{Z}_m$. To find the smallest suitable Fermat ring it is sufficient to choose $m = 2^p + 1 \geqslant kT^{2n/T} > \max_{i=0}^{T-1} c_i$, which is verified by

$$p = 3 \left\lceil \frac{n}{T^2} \right\rceil T \qquad \text{for} \quad n \geqslant 16.$$

[2] We shall discuss later the choice of the upper extreme of this interval.

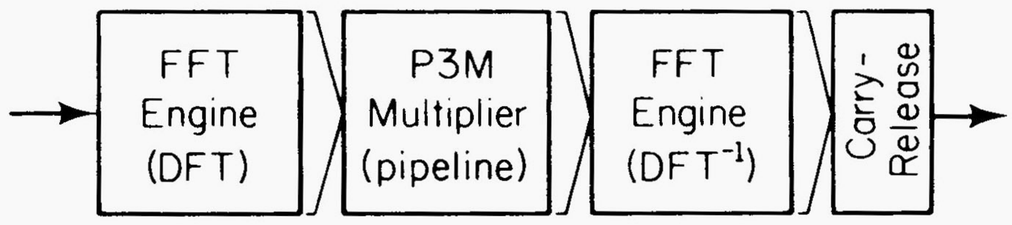


FIG. 3. General scheme of the multiplier.

Therefore, if we select

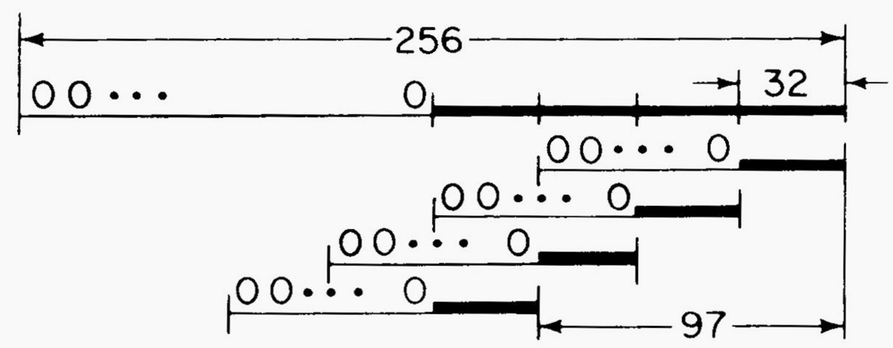
$$p = 3\left\lceil \frac{n}{T^2} \right\rceil T,$$

$$m = 2^p + 1,$$

$$\omega = 2^{2p/T} = 2^{6\lceil n/T^2 \rceil},$$

we verify the hypotheses of Proposition 1 with $r = T$, so that ω is a primitive root of unity in $\mathbf{Z}_m$ of order T.

EXAMPLE. For $n = 256 = 2^8$, and $T = 8$, we have $p = 96$ and $\omega = 2^{24}$. A 128-bit operand is subdivided as illustrated below. Each of the four "chunks"



is further embedded into a 97-bit string, as a member of $\mathbf{Z}_{2^{96}+1}$.

Each transfer of data from major subnetwork to major subnetwork (see Fig. 3), as well as from the input and to the output, involves $O(n/T)$ bits at a time: specifically, $(p+1)$ bits between modules and n/T bits in I/O transfers. Thus each transfer uses $O(T)$ time.

The pipelined multiplier is a straightforward variant of a pipelined 3-multiplication multiplier: it uses length-$(p+1)$ operands and has area $O(n^2/T^2)$, since $p = O(n/T)$. Due to its pipeline structure, it performs multiplications in time $O(T + \log(n/T)) = O(T)$ since $T \geqslant \log n$. Thus, the pipelined multiplier subnetwork has area and time obeying the $AT^2 = \theta(n^2)$ target.

The FFT-engines (both for the direct DFT and for its inverse) are the crucial components of the network and will now be described in some detail. Each consists of T *macromodules* organized as a $\lfloor\sqrt{T}\rfloor \times \lceil\sqrt{T}\rceil$ mesh (see Fig. 4, where we have implicitly assumed that T is a square). If the length of each side of the macromodule is $O(n/T^{3/2})$, then each DFT-engine has area $O(n^2/T^2)$, sufficient to achieve AT^2-optimality. Next we shall show that this objective is attainable.

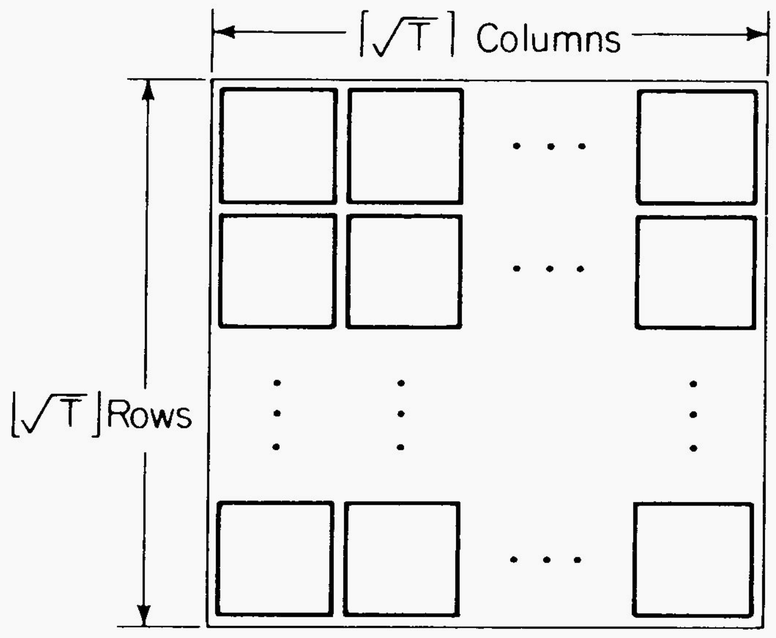


FIG. 4. Architecture of the FFT-engine.

It has been shown in Preparata (1983)[3] how a mesh-connected architecture of $s \times s$ modules can be used to compute the FFT of s^2 elements in $O(s)$ "parallel exchange steps" and $O(\log s)$ "parallel butterfly steps," where an exchange step involves the exchange of the operands of two adjacent modules and the butterfly involves a multiplication by a power ω^i of the principal root and an addition–subtraction. With this background, each macromodule of the mesh is designed to contain a $\mathbf{Z}_m$-operand represented in redundant radix-4 form (i.e., with $3(p/2+1)$ bits) and must have the following capabilities:

1. Transfer its operand to an adjacent module (or exchange operands with an adjacent module);

2. Add two operands (or, equivalently in the redundant representation, subtract one from the other);

3. Multiply an operand by ω^i $(i = 0,..., -1)$.

As noted earlier, we have $O(\sqrt{T})$ operations of type 1 and $O(\log T)$ operations of types 2 and 3; thus, since T is our target computation time, the target times are $O(\sqrt{T})$ and $O(T/\log T)$ for the two types of operations, respectively.

The macromodule, which is designed to store an $O(n/T)$-bit operand, will be structured as follows. It contains $n/T^{3/2}$ $\theta(\sqrt{T})$-bit shift registers, as illustrated in Fig. 5. The length of either side of the macromodule square is $O(n/T^{3/2} + T^{1/2}) = O(n/T^{3/2})$ since $T \leqslant \sqrt{n}$, thus attaining the desired area. Note that the shift-registers can always be arranged as illustrated, since the register length is of lower order than the length of the macromodule side for all T. It is also straightforward to conclude that time $\theta(\sqrt{T})$ for an exchange operation is achieved by the proposed structure, by shifting in parallel the content of each shift-register to the homologous shift-register in one of the adjacent macromodules.

[3] For convenience, see Appendix 2 for a review of the technique.

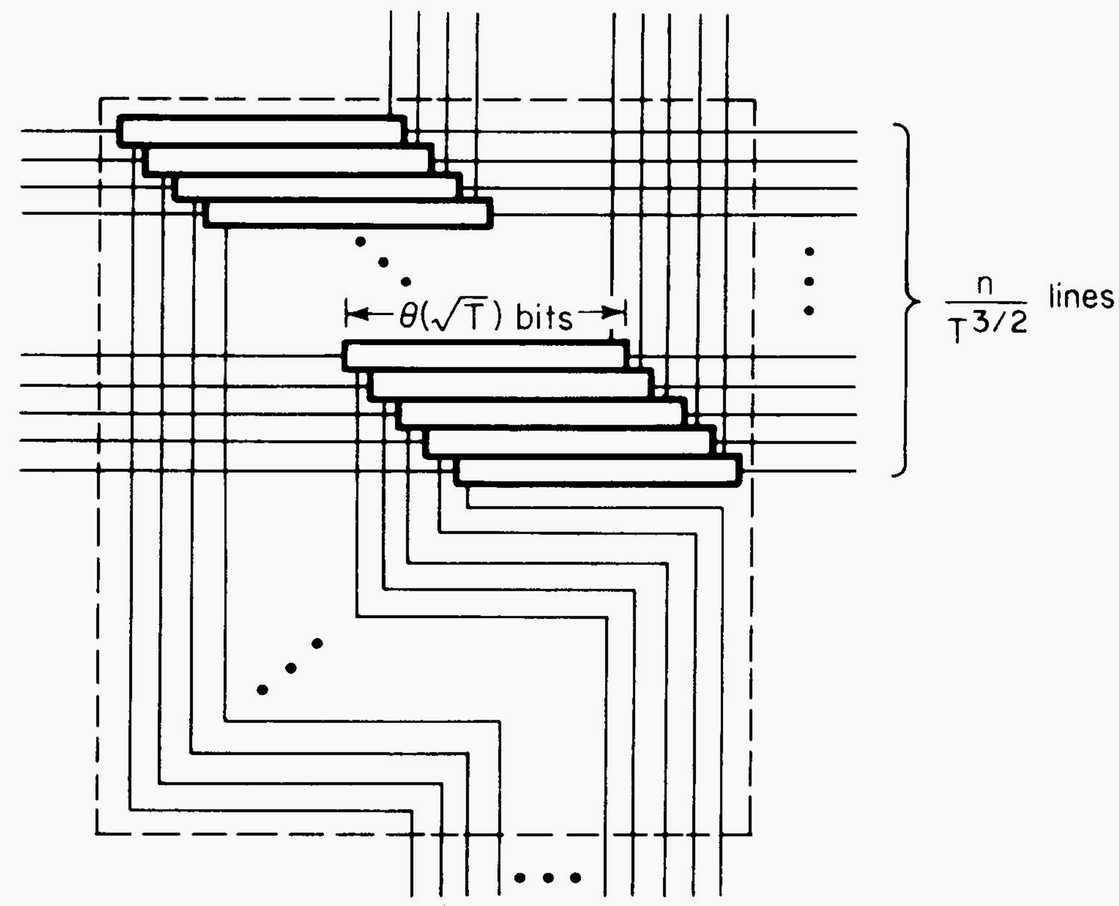


FIG. 5. Structure of a macromodule.

The structure obtained so far is also quite adequate for the execution of type 2 operations (additions–subtractions), by simply equipping each register with a serial adder and introducing a few extra wires to transmit the carries between registers and to perform R-normalization, as defined in Appendix 1.

More delicate is the implementation of type 3 operations. Since multiplication by $\omega^i = 2^{6\lceil n/T^2 \rceil i}$ is basically a left cyclic shift by $O(\lceil n/T^2 \rceil i)$ positions (for $i = 0, 1, \ldots, T-1$), we must provide an interconnection capable of performing any one of T different cyclic shifts of data blocks of size $O(n/T^2)$. Thus the basic information unit dealt with in type 3 operations is a block of $O(n/T^2)$ bits, which we stipulate to be stored in a *micromodule*. (Thus a macromodule consists of T micromodules.) For the sake of simplicity, we assume temporarily that $T \leqslant C_2 n^{2/5}$, for some constant C_2. With this hypothesis each micromodule is an assembly of $O(n/T^{5/2})$ continguous registers. (C_2 is chosen so that this number of registers is at least 1.) The transfer of the content of a micromodule occurs, with a bandwidth equal to the number of its constituent registers, in time $O(\sqrt{T})$.

To perform the desired cyclic shift, we propose to interconnect the T micromodules as an appropriate cube-connected-cycles (CCC) network.[4] Specifically, we shall realize a CCC of 2^{v-u} cycles, each cycle consisting of 2^u micromodules (referred to as a $2^u \times 2^{v-u}$ CCC), where $v = \log_2 T$ and $u = \lceil \frac{1}{2} \log_2 T - \log_2 \log_2 T + c \rceil$, for a suitable constant c. Since it is a functional requirement of the CCC that $2^u \geqslant v - u$ (see Appendix 3), we have the condition

$$2^{\lceil \log_2 T - \log_2 \log_2 T + c \rceil / 2} \geqslant \log_2 T - \lceil \tfrac{1}{2} \log_2 T \log_2 \log_2 T + c \rceil.$$

[4] In Appendix 3 the reader will find a concise description of the CCC.

It can be easily verified that this constraint is always satisfied for $T \geqslant 4$ by choosing $c \simeq 1.308$. Note that this CCC has cycles of length $O(\sqrt{T}/\log T)$ and is capable of performing any of the T prescribed cyclic shifts in a number of steps also $O(\sqrt{T}/\log T)$. Since, as noted earlier, the total available time for a cyclic shift is $O(T/\log T)$, the time available for each CCC-step is $O(\sqrt{T})$, which is exactly the time used to transfer the content of a micromodule. Thus a CCC interconnection realizes the desired computation time for type 3 operations.

We must still verify that the described CCC can be embedded in the macromodule with an insignificant increase in the area (i.e., an area blowup by a constant factor). The modified layout is obtained by dilating, by a factor of 2, each side of the original layout. The new tracks made available are used to realize the CCC connections: specifically, the upper-right portion is used for the cycle links, whereas the lower-left portion is used for the lateral connections. The scheme is illustrated for a 4×4 CCC in Fig. 6 and requires no further comment.

The same considerations apply to the FFT engine designed to implement the inverse FFT. (The only additional operations are the multiplication of each result by $1/T = -2^p/T$—the negative of a power of two (see Appendix 1).)

Remark. We now consider the case $T > C_2 n^{2/5}$. In this situation each shift register contains $O(T/n^{2/5})$ micromodules, since a shift register holds $\theta(T^{1/2})$ bits and a micromodule holds $\theta(n/T^2)$ bits. Also, as before, there are $n/T^{3/2}$ shift registers per macromodule.

Within each macromodule we realize a CCC *whose nodes are now the registers* (not the micromodules as before). The CCC has 2^{v-u} cycles, each cycle consisting of 2^u registers, where $v = \log_2(n/T^{3/2})$ and $u = \log_2(T^{1/2}/\log T)$. In particular, the cycle length is $T^{1/2}/\log T$. We can clearly embed the CCC with only a constant blowup in area. The interconnection of

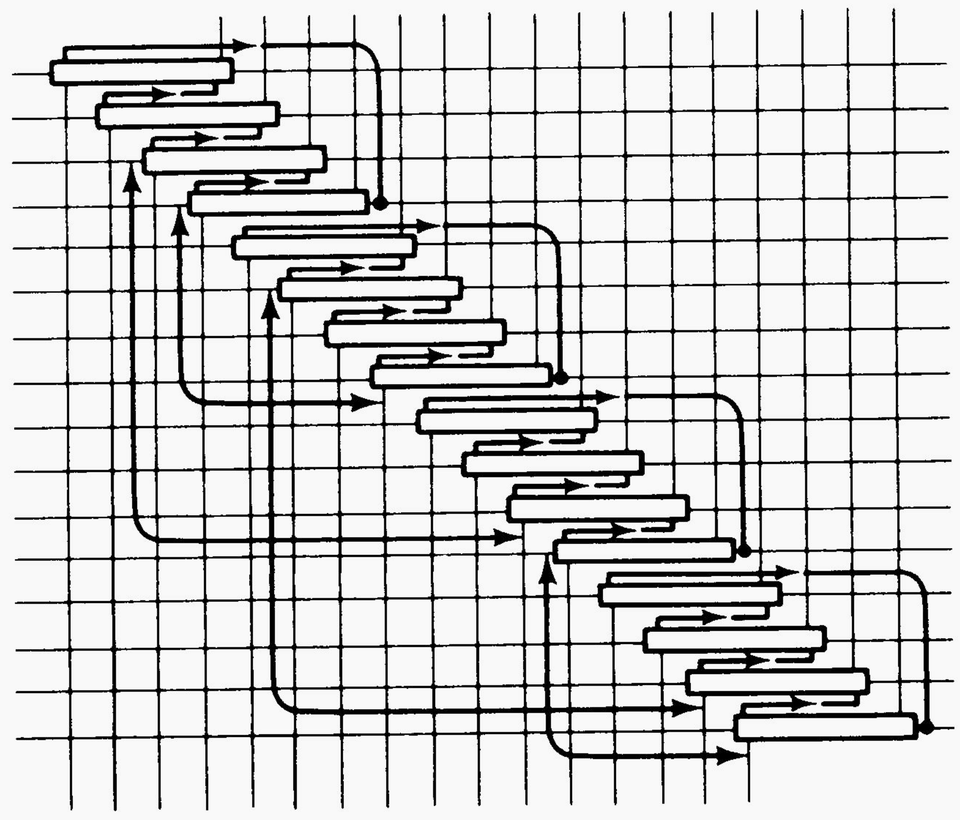


FIG. 6. Embedding of a 4×4 CCC into a 16-element macromodule.

the registers is completed by (switchable) links that have the capacity of reconfiguring the registers into a single (N/T)-bit cyclic shift register.

Consider now a cyclic shift by $(n/T^2)i$ positions, $0 \leqslant i < T$, and write

$$(n/T^2)i = aT^{1/2} + b,$$

where a and b are integers with $b < T^{1/2}$. We can realize the cyclic shift of the content of a macromodule by $(n/T^2)i$ positions in two steps:

(1) Shift by b positions. This can clearly be done in time $O(b) = O(T^{1/2})$ by using the registers reconfigured as a single shift register.

(2) Shift by $a \cdot T^{1/2}$ positions. Since $aT^{1/2}$ is a multiple of the shift register length we can perform such a shift using the CCC. It takes $O(T^{1/2}/\log T)$ CCC-steps, and hence $O(T/\log T)$ time units. (Recall that shift registers are transferred bit-serially.)
We conclude that the total time needed for the shift is $O(T/\log T)$.

Since time $O(T/\log T)$ is available for each shift, as noted earlier, we conclude that we are operating within the desired time bound.

One final comment is in order with regard to the transfer of data from the DFT-engine to the pipeline multiplier (and vice versa). Elements of $\mathbf{Z}_m$ have to be transferred in parallel on the entire channel of bandwidth $O(n/T)$, whereas at the completion of the DFT computation each such element is wholly stored in a macromodule. Therefore a preliminary data rearrangement is necessary, that will bring all microoperands of a given macromodule to be aligned (in a given row or column). The data paths necessary for this rearrangement are available, and it is left as an exercise to show that $O(\sqrt{T})$ time suffices to complete this task.

Since all major modules of Fig. 3 have area $O(n^2/T^2)$, and the time used for the DFTs and the pointwise multiplications is $O(T)$ (notice that this time adequately accounts for the release-of-the-carries and the conversions in $\mathbf{Z}_m$ to nonredundant form), we have the followong conclusions:

THEOREM. *It is possible to construct VLSI multipliers of n-bit numbers with the optimal performance* $AT^2 = \theta(n^2)$ *for all computation times T such that* $\Omega(\log n) \leqslant T \leqslant O(\sqrt{n})$.

APPENDIX 1: THE ARITHMETIC OF FERMAT RINGS

The operands considered in this paper are elements of a Fermat ring[5] $\mathbf{Z}_m$ of the integers modulo $m = 2^p + 1$, where p is an even integer (to be chosen later). The operands are also represented in a redundant radix-4 form, where

[5] Fermat rings were used by Schönhage and Strassen (1971) in their fast multiplication technique.

$$
\begin{array}{l}
a_{L-1} \cdots a_i \cdots \quad a_1\ a_0\ + \\
b_{L-1} \cdots b_i \cdots \quad b_1\ b_0 \\
\hline
s^*_{L-1} \cdots s^*_i \quad \cdots \quad s^*_1\ s^*_0 \\
c_{L-1} \cdots c_{1+1} \cdots c_2\ c_1\ 0
\end{array}
$$

FIG. 7. Illustration of the first step of addition for numbers in redundant radix-4 representation.

the digits belong to the set $\{-3, -2, -1, 0, 1, 2, 3\}$. Thus the value of a digit string $(a_{L-1}, a_{L-2}, \ldots, a_0)$, with $L = p/2 + 1$, is

$$\sum_{i=0}^{L-1} a_i 4^i,$$

which yields an operand range $R \triangleq [-4^L + 1, 4^L - 1] \supseteq \mathbf{Z}_m$. (Notice also that $4m > 4^L - 1$.) We shall call *R-normalization* the operation of bringing a number within the range R modulo m, i.e., to go from $x \in \mathbf{Z}$ to $y \in R$ with $y = x \pmod m$.

We shall discuss the operations of addition/subtraction, multiplication and division by a power of 4, and conversion between redundant and irredundant forms.

(i) *Addition–subtraction in* $\mathbf{Z}_m$. Since $a = \sum_{i=0}^{L-1} a_i 4^i$ means $-a = \sum_{i=0}^{L-1} (-a_i) 4^i$, subtraction reduces trivially to addition. Suppose then we wish to add modulo m the two numbers in R

$$a = \sum_{i=0}^{L-1} a_i 4^i \qquad \text{and} \qquad b = \sum_{i=0}^{L-1} b_i 4^i$$

so that their sum is also normalized in R. Referring to Fig. 7, for each $i = 0, \ldots, L-1$, we first compute the digit pair (s_i^*, c_{i+1}) from the pair (a_i, b_i), according to Table I. Notice that $a_i + b_i = s_i^* + 4c_{i+1}$, $s_i^* \in \{-2, -1, 0, 1, 2\}$ and $c_{i+1} \in \{-1, 0, 1\}$. To obtain the final sum, we distinguish various cases:

TABLE I

$a_i + b_i$	−6	−5	−4	−3	−2	−1	0	1	2	3	4	5	6
s_i^*	−2	−1	0	1	−2	−1	0	1	2	−1	0	1	2
c_{i+1}	−1	−1	−1	−1	0	0	0	0	0	1	1	1	1

1. $c_L = 0$. In this case, the integer represented by

$$(s_{L-1},\ldots, s_0), \qquad s_i = s_i^* + c_i \qquad (-3 \leqslant s_i \leqslant 3,\ i = 0,\ldots, L-1)$$

is a legitimate representation of the sum $(a + b) \bmod m$ in the range R.

2. $c_L \neq 0$. In this case the result does not belong to R, so that a correction is necessary to accomplish R-normalization. Specifically, we have

$$(a+b) \bmod m = \left(\sum_{j=0}^{L-1} s_j 4^j + c_L 4^L \bmod m \right) \bmod m$$

$$= \left(\sum_{j=0}^{L-1} s_j 4^j - 4c_L \right) \bmod m$$

since $4^L = 2^{p+2} = -4 \bmod m$. We further distinguish two subcases:

2.1. $c_L s_1 \neq -3$. In this case the final sum is obtained by simply replacing s_1 with $s_1' = s_1 - c_L$, since $-3 \leqslant s_1 - c_L \leqslant -3$.

2.2. $c_L s_1 = -3$. In this case $s_1 - c_L = -4$ or 4, so the correction of case 2.1 is not applicable. We then apply the same technique to perform the addition $(S + C) \bmod m$, where $S = \sum_{j=0}^{L-1} s_j 4^j$ and $C = -4c_L$. Letting $s_1 - c_L = s_1^{**} + c_2'$, we note that $s_1^{**} = 0$ (since $s_1 - c_L = -4$ or 4), whence in forming the final sum case 2.2 cannot arise again.

It follows from the preceding discussion that addition can be done in $O(1)$ time.

(ii) *Multiplication and division by a power of* 4. Let $a = \sum_{j=0}^{L-1} a_j 4^j$ and consider the product $a \cdot 4^s \bmod m$, for some integer s. We have

$$a4^s \bmod m = \sum_{j=0}^{L-1} a_j 4^{j+s} \bmod m$$

$$= \left(\sum_{h=s}^{L-1} a_{h-s} 4^h + \sum_{h=L}^{L-1+s} a_{h-s} 4^h \bmod m \right) \bmod m$$

$$= \left(\sum_{h=s}^{L-1} a_{h-s} 4^h + \sum_{i=0}^{s-1} a_{L-s+i} 4^i (-4) \right) \bmod m$$

since $4^L \bmod m = -4$. Thus multiplication by 4^s is equivalent to:

(a) cyclically shifting to the left the L-digit string by s digit positions;

(b) changing the sign of the s least significant digits of the string obtained in (a) and shifting them one position to the left;

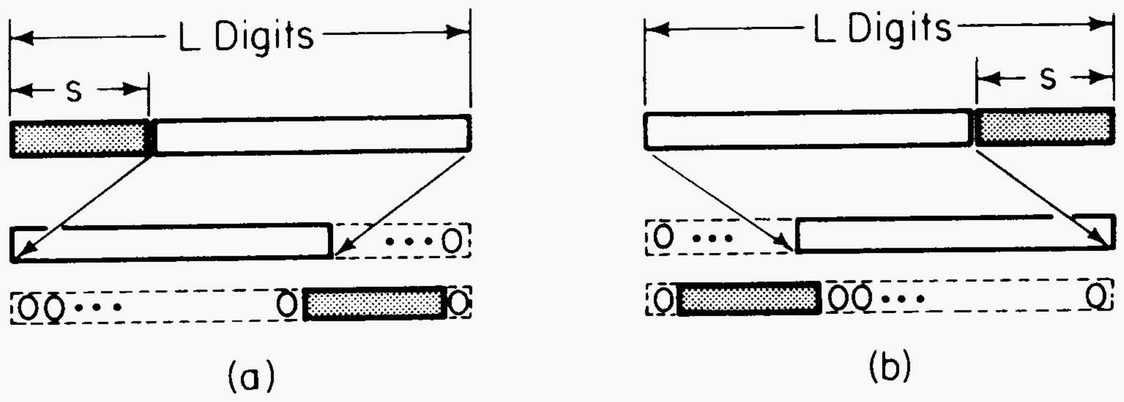


FIG. 8. Illustration of multiplication and division by a power of 4.

(c) adding the two resulting numbers with the method described above.

These opereations are illustrated schematically in Fig. 8a.

With regard to divisions by powers of 4, we know that any power of two 2^s $(s \leqslant p)$ has a multiplicative inverse in $\mathbf{Z}_m$, given by $2^p + 1 - 2^{p-s}$. The inverse of a power of two in $\mathbf{Z}_m$, however, is not a power of two, and so multiplication by it does not exhibit the interesting feature described above. However, since we chose to represent the elements of $\mathbf{Z}_m$ in R, we represent the multiplicative inverse of 2^s as -2^{p-s}, so that multiplication by it becomes a right cyclic shift by s digit positions, with subsequent negation of the s most significant digits and their shift one further position to the right, as shown in Fig. 8b.

Since sign changes and additions are performed in constant time, the computation time is dominated by the time used by the shift operations.

(iii) *Multiplications in* $\mathbf{Z}_m$. Whatever multiplication scheme we adopt (see Section 2), the result p is a $2L$-digit number. To bring it within range R (L-digit numbers), we operate as follows:

$$
\begin{aligned}
p = \sum_{i=0}^{2L-1} p_i 4^i &= \sum_{i=0}^{L-1} p_i 4^i + \sum_{i=0}^{L-1} p_{L+i} 4^{L+i} \\
&= \sum_{i=0}^{L-1} p_i 4^i - \sum_{i=0}^{L-1} p_{L+i} 4^i \qquad \bmod m \\
&= \sum_{i=0}^{L-1} p_i 4^i - \sum_{i=0}^{L-1} p_{L-1+i} 4^i + 4p_{2L-1} \qquad \bmod m.
\end{aligned}
$$

Thus, to perform the R-normalization of the product we must perform a shift and two additions over $\mathbf{Z}_m$.

(iv) *Conversion between binary form and redundant radix-4 form.* Without loss of generality, we assume that the input binary numbers are nonnegative and represented in 2's complement form with $p+2$ bits $b_{p+1}, \ldots, b_0$ ($b_{p+1} = 0$ is the sign bit). The conversion to radix-4 form is

trivially accomplished by inserting 0 to the left of b_{2i+1} for $i = 0, 1, \ldots, p/2$, in constant time.

The conversion to binary form is somewhat more complex. One possible implementation consists of forming two redundant radix-4 numbers a_+ and a_-, consisting respectively of the positive and negative digits of the given number a. Next, each digit of a_+ and a_- is converted to the binary representation of its modulus, thereby obtaining two binary numbers a'_+ and a'_-; all of these operations take time $O(1)$.

Finally we compute $a^* = a'_+ - a'_-$ by a binary subtraction (in time $O(\log p)$. Notice that a^* belongs to R but not necessarily to $\mathbf{Z}_m$: so, to compute $a^* \bmod m$ ($\mathbf{Z}_m$-normalization) it may be necessary to add/subtract m at most four times, since $4m > 4^L - 1$.

APPENDIX 2: A MESH-CONNECTED NETWORK FOR THE DISCRETE FOURIER TRANSFORM[6]

Let G be a commutative ring containing a primitive root of unity, ω, of order $k = m^2$ in G. We then have the following two facts:

A1. The DFT $\langle A_0, A_1, \ldots, A_{k-1} \rangle$ of a vector $\langle a_0, a_1, \ldots, a_{k-1} \rangle$ can be obtained as a two-dimensional DFT, by arranging the vector in row-major order as $m \times m$ matrix $A = \|a_{ij}\|$, where $a_{ij} = a_{mi+j}$ $(j < m)$. (Note that indexing starts from 0 rather than from 1.) Letting $A_{rs} = A_{mr+s}$, we then have

$$A_{rs} = \sum_{ij} a_{ij} \omega^{(mi+j)(mr+s)} = \sum_{j=0}^{m-1} (\omega^m)^{jr} \left(\omega^{sj} \sum_{i=0}^{m-1} a_{ij} (\omega^m)^{is} \right). \qquad (1)$$

The latter expression suggests the following algorithm

D1. $A'_{sj} \leftarrow \sum_{i=0}^{m-1} a_{ij} (\omega^m)^{is}$ (DFT of each column of the matrix);

D2. $A''_{sj} \leftarrow \omega^{sj} A'_{sj}$ (local multiplication);

D3. $A'''_{sr} \leftarrow \sum_{j=0}^{m-1} A''_{sj} (\omega^m)^{jr}$ (DFT of each row of the matrix).

(Note that $A'''_{sr} = A_{rs}$; i.e., the algorithm obtains in reality the transpose of the desired matrix.) This method has already been used in Brent and Kung (1981), where, however, the DFT itself was obtained through matrix multiplication.

[6] Preparata (1983).

A2. A unidimensional m-module array (where $m = 2^r$ for convenience) can be used to compute the DFT of an m-vector, as has been shown in Preparata and Vuillemin (1981a, b). This computation uses $\theta(m)$ exchange steps and $\theta(\log m)$ "butterfly" steps.

Thus, if we have an $m \times m$ mesh of k modules, the columns of the mesh are first used to execute in parallel Step D1 according to the scheme alluded to in A2, and—following the local multiplication D2—the rows of the mesh are finally used to execute in parallel Step D3 (again using the scheme A2).

APPENDIX 3: STRUCTURE AND OPERATION OF THE CUBE-CONNECTED-CYCLES NETWORK

The $2^u \times 2^{v-u}$ cube-connected-cycles (CCC)[7] is a network of 2^v modules, which can be conveniently thought of as a $2^u \times 2^{v-u}$ array of processors $P[i,j]$ $(0 \leqslant i < 2^u,\ 0 \leqslant j < 2^{v-u})$, arranged as a matrix where j grows from left to right and i grows from bottom to top. The CCC-processor $P[i,j]$ has *number* $h = j \cdot 2^u + i$. The columns of the $2^u \times 2^{v-u}$ arrays are connected as cycles; i.e., there is a connection between $P[i,j]$ and $P[(i+1) \bmod 2^u, j]$ for $0 \leqslant i < 2^u,\ 0 \leqslant j < 2^{v-u}$. Furthermore, there is a link between processors $P[i,j]$ and $P[i,j']$, i.e., processors in the same row, if the binary representations of j and j' differ *exactly* in bit position i; these links are called *lateral connections*. A 4×4 CCC is shown in Fig. 9.

It has been shown[7] that a 2^v-processor CCC emulates the v-dimensional binary cube architecture, in executing the algorithms that requires the successive use of the cube dimensions $\{E_0, \ldots, E_{v-1}\}$, either in the order $E_0, \ldots, E_{v-1}$ (ASCEND) or in the reverse order (DESCEND). (Such an algorithmic paradigm has been referred to as "recursive combination.") In more detail, and referring for concreteness to the *ASCEND* schedule, the CCC cycles emulate cube dimensions $E_0, E_1, \ldots, E_{u-1}$ (cycle dimensions), whereas the lateral connections are used to emulate cube dimensions E_u, $E_{u+1}, \ldots, E_{v-1}$ (lateral dimensions). It is therefore clear that, due to the assignment of rows to dimensions, a cycle must contain at least as many processors as there are lateral dimensions; that is,

$$2^u \geqslant v - u.$$

The time used by the CCC to carry out an *ASCEND* or *DESCEND* algorithm is proportional to the CCC cycle length.

The operation of cyclic shift has been shown to be a representative of the recursive combination paradigm, and therefore can be executed by the CCC.

[7] Preparata and Vuillemin (1981a).

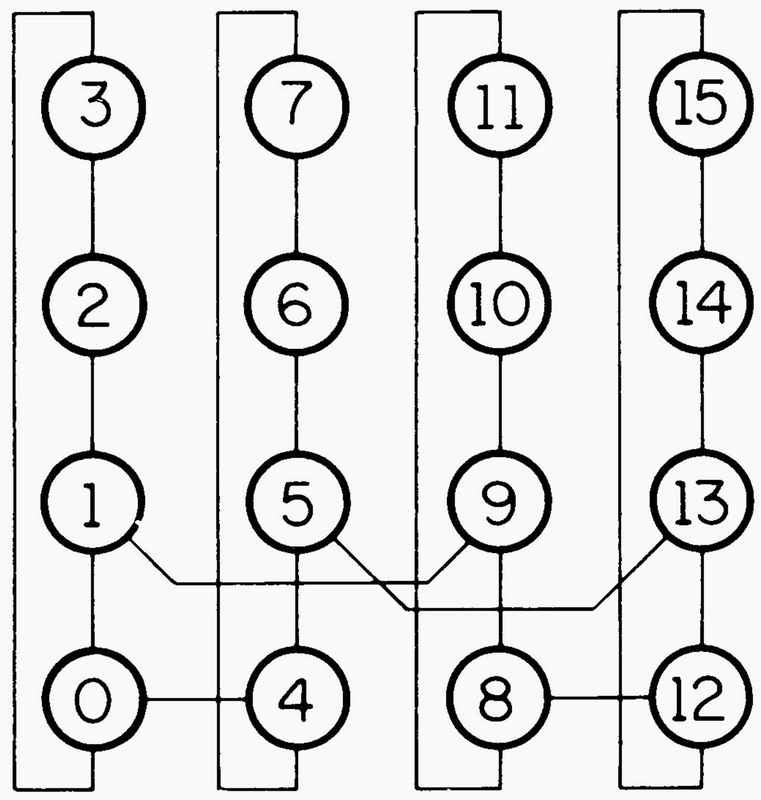


FIG. 9. A 4×4 CCC. Processors are labelled with their numbers ($v = 4$, $u = 2$).

RECEIVED August 11, 1983; ACCEPTED December 21, 1983

REFERENCES

ABELSON, H. AND ANDREAE, P. (1980), Information transfer and area-time trade-offs for VLSI multiplication, *Comm. ACM* **23**, No. 1, 20–22.

AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. (1974), "The Design and Analysis of Computer Algorithms," Addison–Wesley, Reading, Mass.

BECKER, B. "Schnelle Multiplizierwerke für VLSI—Implementierung," Technical Report, Uni. des Saarlandes, 1982.

BRENT, R. P., AND KUNG, H. T. (1981), The chip complexity of binary arithmetic, *J. Assoc. Comput. Mach.* **28**, 521–534.

DADDA, L. (1965), Some schemes for parallel multipliers, *Alta Frequenza* **34**, 343–356.

KARAZUBA, A AND OFMAN, Y. (1962), Multiplication of multidigit numbers on automata, *Dokl. Akad. Nauk SSSR* **145**, 293–294.

LENGAUER, T., AND MEHLHORN, K. (1983), VLSI complexity theory, efficient VLSI algorithms and the HILL design system, *in* "The International Professorship in Computer Science: Algorithmics for VLSI" (Trullemans, Ed.), Academic Press, New York, in press.

LUK, W. K., AND VUILLEMIN, J. E. (1983), "Recursive Implementation of Optimal Time VLSI Integer Multipliers," VLSI 83, Trondheim, Norway, September.

MULLER, D. E. (1963), Asynchronous logic and application to information processing, *in* "Switching Theory in Space Technology" (Aiken and Main, Eds.), Stanford Univ. Press, Stanford, Calif.

PREPARATA, F. P. (1983), An area-time optimal mesh-connected multiplier of large integers, *IEEE. Trans. Comput.* **C-32**, No. 2, 194–198.

PREPARATA, F. P., AND VUILLEMIN, J. (1981a) The cube-connected-cycles: A versatile network for parallel computation, *Comm. ACM* **24**, No. 5, 300–309.

PREPARATA, F. P., AND VUILLEMIN, J. (1981b), Area-time optimal VLSI networks for computing integer multiplication and discrete Fourier transform, *in* "Proceedings, I.C.A.L.P., Haifa, Israel," pp. 29–40.

SCHÖNHAGE, A., AND STRASSEN, V. (1971), Schnelle Multiplikation grosser Zahlen, *Computing* **7**, 281–292.

THOMPSON, C. D. (1979), Area-time complexity for VLSI, *in* "Proceedings, 11th Annual ACM Symposium on the Theory of Computing (SIGACT)," pp. 81–88.

VUILLEMIN, J. E. (1983), A very fast multiplication algorithm for VLSI implementation, *Integration, VLSI J.* **1**, No. 1, 33–52.

WALLACE, C. S. (1964), A suggestion for a fast multiplier, *IEEE Trans. Comput.* **EC-13**, No. 2, 14–17.

Printed by the St. Catherine Press Ltd., Tempelhof 41, Bruges, Belgium