The Chip Complexity of Binary Arithmetic

R. P. Brent
Department of Computer Science
Australian National University
Canberra, A. C. T. 2600
Australia

H. T. Kung
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213
U. S. A.

1. Introduction

The chip complexity of a computation is concerned with the chip area, A, and the time, T, required to perform the computation when implemented on a chip. An area-time product AT^{α}, for $\alpha \geq 0$, is used as a complexity measure. A particular value of α, which is chosen by the user, reflects the relative importance between A and T. This paper derives lower and upper bounds on the area-time complexity for chips that implement binary arithmetic, assuming a model of computation which is intended to approximate current and anticipated LSI or VLSI technology.

In Section 2 we describe our computational model and basic assumptions. Section 3 establishes for any n-bit multiplication chip a general lower bound

$$AT^{2\alpha} = \Omega(n^{1+\alpha}) \qquad (1.1)$$

which is valid for all $\alpha \in [0,1]$. The case $\alpha = 1$ was established independently using a more restrictive model than ours, by [Abelson and Andreae 80] (see also ([Savage and Swamy 78]). In Section 4 we sketch a design for n-bit multiplication that gives

$$AT^{2\alpha} = O(n^{1+\alpha} \lg^{1+2\alpha} n), \qquad (1.2)$$

for all $\alpha \geq 0$. Thus the exponent $1 + \alpha$ of n in (1.1) and (1.2) is tight for $\alpha \in [0,1]$. The chip complexity of binary addition is studied in Section 5. We show that for any $1 \leq w \leq n$, n-bit numbers can be added in time $O(n/w + \lg w)$, using area $O(w \lg w + 1)$, if the input bits from each operand are available w at a time. In Section 6 we compare the chip complexity for binary multiplication and addition; we conclude that multiplication is harder than addition, for all complexity measures $AT^{2\alpha}$, $\alpha \geq 0$.

The results of this paper mainly draw on two papers by the authors ([Brent and Kung 79a, Brent and Kung 79b]).

2. The Computational Model and Basic Assumptions

We assume the existence of circuit elements or "gates" which compute a logical function of two inputs in constant time and occupy at least a constant minimum area. Gates are connected by wires which have constant minimum width (equivalently, the wires must be separated by at least some minimal spacing). Our measure of the cost of a design is the *area* rather than the number of gates required. This is an important difference between our model and earlier models of [Winograd 65], [Brent 70] and others. For motivation and discussion of models similar to ours, see [Thompson 79, Leiserson 80].

For proving the results of this paper, various subsets of the following assumptions A1 through A8 are used. Comments and justification are given following the statement of each assumption.

A1. The computation is performed in a convex planar region R of area A.

Because of heat-dissipation, packing and testing requirements, a two-dimensional planar model is reasonable. The convexity assumption is not restrictive in the sense that almost all existing chips or useful modular designs do have convex

[1]This research was supported in part by the National Science Foundation under Grant MCS 78-236-76 and the Office of Naval Research under Contracts N00014-76-C-0370 and N00014-80-C-0236. Most of this work was carried out at the Australian National University while H. T. Kung was visiting there as a Visiting Fellow during May 1979.

boundaries for packaging or modularity reasons. (The convexity assumption can be removed for part of Theorem 3.1 below by using a different proof.)

A2. Wires have minimal width $\lambda > 0$.

λ is assumed constant, but in applications of our results it will of course depend on the technology. We also assume R has width at least λ in every direction.

A3. At most $\nu \geq 2$ wires can overlap (or intersect) at any point in R.

A chip may consist of ν layers. Wire crossings through different layers are allowed. In fact, transistors are typically formed by cross-overs of wires. Since $\nu \geq 2$, the graph of wires (edges) and gates (nodes) need not be planar in a graph-theoretic sense.

A4. I/O ports each contain a $\lambda \times \lambda$ square and thus have area at least $\rho \geq \lambda^2$. An I/O port can be multiplexed to handle more than one input or output variable.

If R is a complete chip, ρ will be large compared to λ^2. If R is only part of a chip and I/O is to other regions on the chip, ρ could be of order λ^2. We do not require each input (or output) variable to appear in a distinct input (or output) port, as required in [Thompson 79]. So I/O port may be multiplexed as they often are in practice.

A5. A bit requires minimal time $\tau > 0$ to propagate along a wire or to transmit through an I/O port. The time for one gate computation and an arbitrary fan-out of the result is included in τ.

Since dimensions are limited by the minimal wire-width λ and minimal gate area, a minimal propagation time is reasonable. We do not need to assume that the propagation time increases with the length of the wire. With the (small) sizes of chips we now have or anticipate for the future, the propagation time, which is the time needed to charge or discharge a wire, is limited by the wire capacitance rather than the velocity of light. A longer wire will generally have a larger capacitance, and thus require a larger driver to maintain constant propagation time, but the driver area need not exceed a fixed percentage of the wire area, so can be ignored if λ is increased slightly; see [Mead and Conway 80]. Although it would be reasonable to assume bounded fanout, we do not need this assumption for proving lower bounds. When proving upper bounds, we do assume bounded fanout.

A6. The times and locations at which input and output bits are available are fixed and independent of the values of the input bits.

When proving upper bounds in Sections 4 and 5, we further assume that if a_i and a_j are any two bits in an operand such that a_i is more significant than a_j, then a_i is not input (or ourput) to the chip before a_j, but they are allowed to be input (or ourput) to the chip in parallel. This assumption holds for all the arithmetic circuits that the authors know.

A7. Storage for one bit of information takes area at least $\beta > 0$.

β is typically several times larger than λ^2.

A8. Each input bit is available only once.

There is no free memory outside R. If the same input bit is required at different times, it must be stored within R, taking area at least β (see A7).

3. Lower Bound Results for Multiplication

Let $p = p_{2n} \ldots p_1$ be the 2n-bit product of n-bit integers $a = a_n \ldots a_1$ and $b = b_n \ldots b_1$.

3.1 Lower bounds for shifting circuits

When $b = 2^j$, p is a shifted j bits to the left. Thus, any multiplier circuit must also be a shifting circuit capable of performing j-bit shifts for all $0 \leq j \leq n-1$.

Theorem 3.1: Under assumptions A1 to A6 of Section 2, any chip that is capable of performing the shifts as described above must satisfy

$$AT^2 \geq K_1 n^2, \tag{3.1}$$

$$AT \geq K_2 Ln \tag{3.2}$$

where

$$K_1 = 2[\lambda\tau(9 - 4\cdot5^{1/2})/\nu]^2, \tag{3.3}$$
$$K_2 = \lambda\tau(9 - 4\cdot5^{1/2})/(\pi\nu),$$

and L is the perimeter of the chip.

Before proving Theorem 3.1, we need two Lemmas.

Lemma 3.1: For any convex planar figure with area A, perimeter L, diameter D, and chord of length C perpendicular to a chord whose length is the diameter D,

$$A \geq CD/2, \tag{3.4}$$

and

$$A \geq CL/2\pi. \tag{3.5}$$

Proof: The results follow from well-known inequalities for convex figures. For a proof (and a definition of "diameter" etc.) see, for example, [Yaglom and Boltyanskii 61]. ■

Lemma 3.2:

$$\min_{0 \leq r < 1} \max(2r, (1-r)^2/8) = 2(9 - 4\cdot5^{1/2}).$$

Proof: It is easy to verify that the minimum occurs when $16r = (1-r)^2$ and the only root of this equation in [0,1] is $r = 9 - 4\cdot5^{1/2}$ ■

Proof of Theorem 3.1: Consider any chip that can perform j-bit shifts for all $0 \le j \le n-1$. By assumption A1, the chip forms a convex region R. Let D be the diameter of R, and Y a chord of length D.

Let $S = \{p_{2n-1}, \ldots, p_n\}$, and let M be the maximum number of elements of S sharing or multiplexing one output port of the chip. By assumption A4, an I/O port has area at least $\rho \ge \lambda^2$. We represent each I/O port by an infinitesimal point on the port. Based on these representives of I/O ports, we partition the chip by a chord X perpendicular to Y as follows. The chord X divides S into two subsets S_1 and S_2 such that representives of the output ports for elements of S_1 lie on one side of X and those for elements of S_2 lie on the other side of X. (Since representives of I/O ports are of infinitesimal size, we can assume that by an infinitesimal perturbation from the perpendicular to Y, X does not intersect any of them.) By "sliding" the intersection of X and Y along Y, we can arrange that

$$|S_i| \le \lfloor (n + M)/2 \rfloor \tag{3.6}$$

for i = 1 and 2. For notational convenience, we use d to denote $\lfloor (n + M)/2 \rfloor$. When the j-bit shift is performed, p_{i+j} takes the value of a_i. For $d \le i \le n$, the ith row in Table 3-1 indicates the p_i's that take the value of a_i under j-bit shifts for all $n-i \le j \le n-1$.

$a_i \setminus j$	0	1 ⋯	n-d ⋯	n-2	n-1
a_d			p_n ⋯	p_{n+d-2}	p_{n+d-1}
a_{d+1}			⋯	p_{n+d-1}	p_{n+d}
⋮					
a_{n-1}		p_n	⋯	p_{2n-3}	p_{2n-2}
a_n	p_n	p_{n+1}	⋯	p_{2n-2}	p_{2n-1}

Table 3-1: The dependence of the p_i's on the a_i's under various shifts.

Note that in the table all the p_i's belong to S, which is divided into two parts by the chord X. By (3.6), in the ith row of the table there are at most d of the p_i's for which the representives of the output ports lie on the same side of X as the representative of the input port for a_i. Consequently, in the ith row there are at least i-d of the p_i's for which the representative of the output ports do not lie on the same side of X as the representive of the input port for a_i. For all rows in the table, there are a total of at least $\sum_{d \le i \le n} (i - d) \ge (n - M)^2/8$ such p_i's. This implies that one of the n columns in the table, say, the jth column, must have at least $(n - M)^2/8n$ such p_i's. In other words, if

$I = \{i \mid i \in \{d, d+1, \ldots, n\}$ and the representative of the input port for a_i does not lie on the same side of X as that of the output port for $p_{i+j}\}$,

then

$$|I| \ge (n - M)^2/8n.$$

For $i \in I$, the input port for a_i or the output port for p_{i+j} may intersect the chord X, although their representatives do not. Define

$I' = \{i \mid i \in I$, and the chord X intersects the input port for a_i or the output port for p_{i+j}, or both$\}$.

Then

$I\text{-}I' = \{i \mid i \in \{d, d+1, \ldots, n\}$, and the input port for a_i and the output port for p_{i+j} do not intersect X and they lie on different sides of X$\}$.

Consider the computation of the j-bit shift. Note that the j-bit shift, which maps a_i to p_{i+j} for $i = 1, \ldots, n$, is an identity mapping. Hence, before the shift is complete, at least $|I\text{-}I'|$ bits of information about a_i, $i \in I\text{-}I'$, must cross X for computing p_{i+j} for $i \in I\text{-}I'$, and at least $|I'|$ bits of information about a_i, $i \in I'$, must input to or output from some I/O ports intersecting with X for computing p_{i+j} for $i \in I'$. Suppose that the chord X is of length C. Then by assumptions A2, A3 and A4, at most $\nu C/\lambda$ wires or I/O ports cross X. Thus, by assumption A5, the time T to perform the j-bit shift must satisfy the inequality:

$$\begin{aligned}(\nu C/\lambda)(T/\tau) &\ge |I\text{-}I'| + |I'| \\ &= |I| \\ &\ge (n - M)^2/8n,\end{aligned}$$

or

$$T \ge (\lambda\tau/\nu C)n(1-r)^2/8 \tag{3.7}$$

where $r = M/n$. Since M outputs come through one output port, assumption A5 gives

$$T \ge M\tau = \tau nr. \tag{3.8}$$

Suppose $M < n$. Then at least one wire or one I/O port crosses X, and assumptions A2 and A4 give

$$C \ge \lambda. \tag{3.9}$$

By assumption A3, $\nu \ge 2$. Combining this with (3.8) and (3.9) gives

$$T \ge \tau nr = (2C\tau/2C)nr \ge (\lambda\tau/\nu C)n2r \tag{3.10}$$

From (3.7) and (3.10) it follows by Lemma 3.2 that

$$T \ge (2K_0/C)n, \tag{3.11}$$

where

$$K_0 = \lambda\tau(9 - 4\cdot5^{1/2})/\nu,$$

so by (3.4),

$$AT^2 \ge (CD/2)(2K_0/C)^2n^2 \ge 2K_0^2n^2, \tag{3.12}$$

since $D \ge C$. Suppose that $M = n$. Then $r = 1$. Since there is at least one output port, assumption A4 gives $A \ge \rho \ge \lambda^2$, so by (3.8)

$$AT^2 \geq (\lambda\tau n)^2 > 2K_0^2 n^2. \qquad (3.13)$$

Result (3.1) follows from (3.12) and (3.13). Result (3.2) follows in a similar way. If $M < n$, the combining (3.11) with (3.5) gives

$$AT \geq (CL/2\pi)(2K_0/C)n = K_2 Ln. \qquad (3.14)$$

Suppose that $M = n$. Since by assumption A2, R has width at least λ in every direction, we can choose a chord that is of length $C \geq \lambda$ and is perpendicular to Y. By (3.5) and (3.8) with $r = 1$, we have

$$AT \geq (CL/2\pi)(\tau n),$$

which gives

$$AT \geq K_2 Ln.$$

■

Since any circuit that performs integer multiplications must also be able to perform shifts, (3.1) and (3.2) hold for any n-bit multiplication chip.

Result (3.2) can sometimes give useful lower bounds which are based on the I/O characteristics of a multiplication or shifting chip. If at one time the chip inputs or outputs a total of z bits along its boundary, then by assumptions A3 and A4 $L \geq z\lambda/\nu$ and (3.2) gives $AT \geq K_2(\lambda z/\nu)n$. Thus for any multiplication scheme that accepts, say $\Omega(n^{1/2})$ input bits simultaneously along the chip boundary, we know immediately that $AT = \Omega(n^{3/2})$ (cf. the multiplication scheme in Section 4).

Result (3.1) (with a smaller constant for K_1) could have been established by a proof parallel to that used by Thompson [79] for the discrete Fourier transform problem. In fact, using his result that relates the area of a graph to its minimum bisection width, one can derive (3.1) without the convexity assumption in A1. Our proof, above, represents a new approach that incorporates geometric properties of the chip boundary in the lower bound proof. We feel that the extra convexity assumption we make is not restrictive, since most existing chips do have convex boundaries for packaging reasons. Furthermore, we note that the convexity assumption is needed for establishing results such as (3.2) that relate AT to perimeter L. In [Brent and Kung 79c], under a similar convexity assumption, tight lower bounds on the minimum area required to layout complete binary (or t-ary) trees are obtained.

An interesting corollary of Theorem 3.1 is that lower bounds in (3.1) and (3.2) hold for chips that perform floating point additions, for which shifts are needed to equalize exponents. This explains why the area-time requirements for floating point addition are much higher than those for integer addition, as observed in practical implementation. (Charles Leiserson at CMU first pointed out to one of the authors the application of Theorem 3.1 to floating point addition.)

3.2 A lower bound on the area for multiplier circuits

In Theorem 3.1 we gave lower bounds on AT^2 and AT for shifting circuits. Now, using different techniques, we give a lower bound on A for multiplier circuits.

Theorem 3.2: Under assumptions A4, and A6 to A8, any n-bit multiplication must satisfy

$$A \geq A_0 n,$$

where

$$A_0 = (5/6)[\beta\rho/(\beta+\rho)]. \qquad (3.15)$$

Let $\Phi_n = \{ij \mid 0 \leq i < n,\ 0 \leq j < n\}$ be the set of all integers which can be written as a product of two factors, each less than n; and let $\mu(n) = |\Phi_n|$ be the cardinality of Φ_n. For example, $\Phi_4 = \{0, 1, 2, 3, 4, 6, 9\}$ and $\mu(4) = 7$. Before proving Theorem 3.2, we need lower bounds on $\mu(n)$ and a related function,

$$\delta(n) = \lceil \lg \mu(2^n) + 1 - n \rceil / n. \qquad (3.16)$$

Lemma 3.3:

$$\mu(n) \geq \sigma(n),$$

where $\sigma(n) = \sum_{i \in P_{n-1}} i$ and P_{n-1} is the set of prime numbers smaller than n.

Proof: The numbers pj are distinct if $2 \leq p < n$, p prime, and $1 \leq j \leq p$. Thus, the result follows from the definition of $\mu(n)$. ■

Lemma 3.4: For all $n \geq 4$,

$$\mu(n) \geq n^2/(2 \ln n).$$

Proof: Using a slight modification of Theorem 1 and equation (4.13) of [Rosser and Schoenfeld 62], we can show that for all $n \geq 348$,

$$\sigma(n) > n^2/(2 \ln n).$$

Thus, the result for $n \geq 348$ follows from Lemma 3.3. For $4 \leq n \leq 347$, the result may be verified by a straightforward computation. ■

Lemma 3.5: If $\delta(n)$ is defined by (3.16), then for all $n \geq 1$,

$$\delta(n) \geq 5/6.$$

Proof: From Lemma 3.4

$$\delta(n) \geq \lceil n - \lg(n \ln 2) \rceil / n, \qquad (3.17)$$

and it is easy to verify that the right side of (3.17) is at least 5/6 for all $n \geq 18$. (There is equality for $n = 18$ and $n = 24$.) For $1 \leq n \leq 17$, direct computation shows that $\delta(n) \geq 9/10$. ■

We conjecture that

$$\lim_{n\to\infty} [(\mu(n) \lg \lg n)/n^2] = 1,$$

and

$$\delta(n) \geq 9/10$$

for all $n \geq 1$. Empirical data that support this conjecture are presented in [Brent and Kung 79b].

Proof of Theorem 3.2:

If $n = 1$ there is at least one output port, so $A \geq \rho$, and the result holds. Hence, suppose that $n \geq 2$.

Consider the state of the computation just before the last input bit(s) are accepted. Let m be the number of input bits still to be accepted, so $1 \leq m \leq 2n$.

It is easy to show that there are some inputs a and b such that the output bits $p_{2n}, \ldots, p_n$ are not determined by the 2n-m input bits already accepted. Thus, by assumption A6, at most n-1 bits, $p_{n-1}, \ldots, p_1$, have been output.

Suppose that s bits of information are stored in R at this instant. Then we must have by assumption A8

$$\mu(2^n) \leq 2^{m+(n-1)+s},$$

or the circuit could not produce all $\mu(2^n)$ possible outputs, and would fail for certain inputs. Thus

$$m + s \geq \lceil \lg \mu(2^n)+1-n \rceil = n\delta(n).$$

and, from Lemma 3.5,

$$m + s \geq 5n/6. \tag{3.18}$$

By assumption A7,

$$A \geq \beta s. \tag{3.19}$$

Since a port can accept only one bit at a time, the last m bits must be input through m different ports, so assumption A4 gives

$$A \geq \rho m. \tag{3.20}$$

The result follows easily from (3.18), (3.19) and (3.20). ■

3.3 General lower bounds for the chip complexity of binary multiplication

Theorems 3.1 and 3.2 are the extreme cases $\alpha = 1$ and $\alpha = 0$ of the following result.

Theorem 3.3: Under assumptions A1 to A8, any n-bit multiplication chip must satisfy

$$(A/A_0)\,(T/T_0)^{2\alpha} \geq n^{1+\alpha}, \tag{3.21}$$

for all $\alpha \in [0,1]$. Here A_0 is given by (3.15),

$$T_0 = (K_1/A_0)^{1/2},$$

and K_1 is given by (3.3).

Proof: From Theorem 3.1,

$$(A/A_0)\,(T/T_0)^2 \geq n^2,$$

so

$$(A/A_0)^{\alpha}\,(T/T_0)^{2\alpha} \geq n^{2\alpha}. \tag{3.22}$$

From Theorem 3.2, since $\alpha \in [0,1]$,

$$(A/A_0)^{1-\alpha} \geq n^{1-\alpha}. \tag{3.23}$$

Multiplying (3.22) and (3.23) gives the result. ■

The following corollary of Theorem 3.3 seems worth stating separately, for AT is often used as a complexity measure (see, for example, [Mead and Rem 79]).

Corollary 3.1: Under assumptions A1 to A8, any n-bit multiplication chip must satisfy

$$AT \geq K_3 n^{3/2},$$

where

$$K_3 = A_0 T_0 = (A_0 K_1)^{1/2}.$$

4. Upper Bound Results for Multiplication

It is easy to design practical n-bit multipliers with area $A = O(n)$ and time $T = O(n)$, so

$$AT^{2\alpha} = O(n^{1+2\alpha}). \tag{4.1}$$

For example, the "serial pipeline multipliers" typically used in the implementation of digital filters and signal processors achieve these area and time bounds (see, for example, [Jackson et al. 68, Lyon 76]). In this section we sketch the design of a multiplier with $A = O(n \lg n)$ and $T = O(n^{1/2} \lg n)$, giving

$$AT^{2\alpha} = O(n^{1+\alpha} \lg^{1+2\alpha} n), \tag{4.2}$$

which is asymptotically better than (4.1). The design uses the convolution Theorem to compute the product of two integers in a complex way, and consequently its implementation appears to be difficult. Nevertheless, the design is theoretically interesting because it shows that the exponent $1 + \alpha$ of n in Theorem 3.3 is tight. We do not know if there is any practical design having $AT^{2\alpha} = o(n^{1+2\alpha})$ for $\alpha \in [0, 1]$. Straightforward implementations of "fast" algorithms, for example, the Schonhage-Strassen algorithm [Schonhage and Strassen 71], or the "3-2 reduction" algorithm [Ofman 62, Wallace 64] seem to require area at least order n^2.

In the remainder of this section we assume:

(a) $n = k^2$ is a perfect square, and

(b) $a_j = b_j = 0$ if $j > n/2$.

(If not, n may be increased sufficiently without affecting the asymptotic results.) Let p be the smallest prime of the form

$nq + 1$, $q \geq 1$, F_p the finite field of integers mod p. It is known that $\lg p = O(\lg n)$ (see [Linnik 44, Wagstaff 79]), and that F_p has an n-th root of unity w (see [Bonneau 73]). Let $w = u^k$, so w is a k-th root of unity. Note that in any circuit n is fixed, so we are not concerned with the complexity of finding p, u, w etc: they will be encoded into the circuit. In particular, for facilitating arithmetic in F_p, we assume that a $2\lceil \log_2 p \rceil$-bit approximation to $1/p$ is encoded into the circuit.

In Steps 1-5 below, all arithmetic is done in F_p. In Steps 1-3 we compute the discrete Fourier transform $\underline{a}'$ of $(a_1, ..., a_n)$ and $\underline{b}'$ of $(b_1, ..., b_n)$ over F_p, that is,

$$a'_{j+1} = \sum_{i=0}^{n-1} a_{i+1} u^{ij}$$

for $j = 0, ..., n-1$, etc. In Step 4 we multiply the Fourier transforms. In Step 5 we take the inverse transform, and in Step 6 the final result is computed.

Step 1

Let A, B, U, and W be k by k matrices with elements

$$A_{ij} = a_{(i-1)k+j},$$
$$B_{ij} = b_{(i-1)k+j},$$
$$U_{ij} = u^{(i-1)(j-1)},$$
$$W_{ij} = w^{(i-1)(j-1)}.$$

Perform k by k matrix multiplications to compute

$$A' = WA \text{ and } B' = WB,$$

using a "systolic array" of [Kung and Leiserson 79]. All computations are performed in F_p, so each processing element of the systolic array needs to perform multiplication and addition in F_p. Using a serial pipeline multiplier and a serial adder, a multiplication and addition step in F_p requires no more than area $O(\lg p)$ and time $O(\lg p)$. Thus, Step 1 can be done with area $O(n \lg n)$ and time $O(n^{1/2} \lg n)$.

Step 2

Compute $A'' = A' \circ U$ and $B'' = B' \circ U$, where $\circ$ denotes componentwise multiplication.

Step 3

Compute $A''' = A''W$ and $B''' = B''W$ using the same method as for Step 1. It may be shown that A''' and B''' contain the Fourier transforms of $(a_1, ..., a_n)$ and $(b_1, ..., b_n)$, in fact for $1 \leq i, j \leq k$,

$$A'''_{ij} = a'_{(j-1)k+i},$$
$$B'''_{ij} = b'_{(j-1)k+i}.$$

Step 4

Compute $C''' = A''' \circ B'''$.

Step 5

Compute $C = W^{-1}(U' \circ (C'' W^{-1}))$ as in Steps 1-3. Here $U'_{ij} = u^{-(i-1)(j-1)}$. The matrix C represents the inverse Fourier transform of C'''. Define c_i's by

$$C_{ij} = c_{(i-1)k+j}.$$

Then by the convolution Theorem and our assumption (a) above,

$$c_j = a_1 b_j + a_2 b_{j-1} + ... + a_j b_1 \quad \text{for } 1 \leq j \leq n.$$

Thus,

$$\sum_{i=1}^{2n} p_i 2^{i-1} = \sum_{i=1}^{n} c_i 2^{i-1}.$$

Group the terms in the right hand side into $n^{1/2}$ groups so that the c_i's in each row of the matrix C belong to one group. We obtain

$$\sum_{i=1}^{2n} p_i 2^{i-1} = \sum_{i=1}^{k} R_i 2^{(i-1)k}, \tag{4.3}$$

where

$$R_i = \sum_{j=1}^{k} c_{(i-1)k+j} 2^{j-1}.$$

Given that the c_i's are outputs of the systolic array that computes the matrix C, all the R_i's can be formed in area $O(n \lg n)$ and time $O(n^{1/2} \lg n)$, using the upper bound result regarding addition in Section 5. Thus the problem of computing $p_{2n}, ..., p_1$ has been reduced to the problem of summing $k = n^{1/2}$ terms in the right-hand side of Equation (4.3). Hence, the final step in the computation is:

Step 6

Compute $p_{2n}, ..., p_1$ from the R_i's. The p_i's can be computed, $n^{1/2}$ of them at a time, in area $O(n \lg n)$ and time $O(n^{1/2} \lg n)$, again using the result in Section 5.

This completes our outline of the multiplier with area $A = O(n \lg n)$ and time $T = O(n^{1/2} \lg n)$, giving $AT^{2\alpha} = O(n^{1+\alpha} \lg^{1+2\alpha} n)$. For $0 \leq \alpha \leq 1$, the exponent $1+2\alpha$ of $\lg n$ can certainly be reduced by using probably a still more complicated design than the one outlined above, but we do not know what its minimal value is. For $\alpha > 1$, a design based on the "3-2 reduction" algorithm seems to give $AT^{2\alpha} = O(n^2 \lg^{\delta} n)$ for some $\delta > 0$, which is a better upper bound than the one in (4.2).

5. Upper Bound Results for Addition

Let $a_n a_{n-1} \ldots a_1$ and $b_n b_{n-1} \ldots b_1$ be n-bit binary numbers with sum $s_{n+1} s_n \ldots s_1$. The usual method for addition computes the s_i's by

$$c_0 = 0,$$
$$c_i = (a_i \wedge b_i) \vee (a_i \wedge c_{i-1}) \vee (b_i \wedge c_{i-1}),$$

$$s_i = a_i \oplus b_i \oplus c_{i-1}, \; i = 1, \ldots, n,$$
$$s_{n+1} = c_n,$$

where $\oplus$ means the sum mod 2 and c_i is the carry from bit position i.

It is well-known that the c_i's can be determined using the following scheme:

$$c_0 = 0,$$
$$c_i = g_i \vee (p_i \wedge c_{i-1}), \tag{5.1}$$

where

$$g_i = a_i \wedge b_i,$$

and

$$p_i = a_i \oplus b_i,$$

for i = 1, 2, ..., n. One can view the g_i and p_i as the *carry generate* and *carry propagate* conditions at bit position i. The relation (5.1) corresponds to the fact that the carry c_i is either generated by a_i and b_i or propagated from the previous carry c_{i-1}. This is illustrated in Figure 5-1.

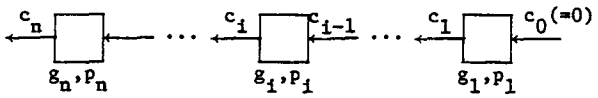

Figure 5-1: The carry chain.

In this section we present a regular and area-efficient layout design for computing all the carries in parallel assuming that the g_i's and p_i's are given. Using this layout design for the carry computation, one can design a parallel adder in a straightforward way (see [Brent and Kung 79a]). The basis of our method is the reduction of carry computation to a *prefix* computation, as described in the following subsection. Although the same idea was used by [Ladner and Fischer 77], their results are not directly applicable because they ignored fanout restrictions, and used the gate count rather than area as a complexity measure.

5.1 Reformulation of the Carry Chain Computation

We define an operator "o" as follows:

$$(g, p) \text{ o } (g', p') = (g \vee (p \wedge g'), p \wedge p'),$$

for any Boolean variables g, p, g' and p'.

Lemma 5.1: Let

$$(G_i, P_i) = \begin{cases} (g_1, p_1) \text{ if } i = 1, \\ (g_i, p_i) \text{ o } (G_{i-1}, P_{i-1}) \text{ if } 2 \le i \le n. \end{cases}$$

Then

$$c_i = G_i \quad \text{for } i = 1, 2, \ldots, n.$$

Proof:

We prove the Lemma by induction on i. Since $c_0 = 0$, (5.1) gives

$$c_1 = g_1 \vee (p_1 \wedge 0) = g_1 = G_1,$$

so the result holds for i = 1. If i > 1 and $c_{i-1} = G_{i-1}$, then

$$\begin{aligned} (G_i, P_i) &= (g_i, p_i) \text{ o } (G_{i-1}, P_{i-1}) \\ &= (g_i, p_i) \text{ o } (c_{i-1}, P_{i-1}) \\ &= (g_i \vee (p_i \wedge c_{i-1}), p_i \wedge P_{i-1}). \end{aligned}$$

Thus

$$G_i = g_i \vee (p_i \wedge c_{i-1})$$

and, from (5.1), we have

$$G_i = c_i.$$

The result now follows by induction. ■

From the definition of the operator o, it is straightforward to show that the operator is associative. Thus, by Lemma 5.1, to compute c_i's it suffices to compute all the (G_i, P_i)'s, and

$$(G_i, P_i) = (g_i, p_i) \text{ o } (g_{i-1}, p_{i-1}) \text{ o } \ldots \text{ o } (g_1, p_1)$$

can be evaluated in any order from the given g_i's and p_i's. (Intuitively, G_i may be regarded as a "block carry generate" condition, and P_i as a "block carry propagate" condition.)

5.2 A layout for the carry chain computation

Consider first the simpler problem of computing the (G_i, P_i) for i = n only. Since the operator "o" is associative, (G_n, P_n) can be computed in the order defined by a binary tree. This is illustrated in Figure 5-2 for the case n = 16. In the figure, each black processor performs the function defined by the operator "o" and each white processor simply transmits data. The white and black processors are depicted in Figure 5-3.

Note that for Figure 5-2 each processor is required to produce only one of its two identical outputs, and the units of time are such that one computation by a black processor and propagation of the results takes unit time.

Consider now the general problem of computing the (G_i, P_i) for all $1 \le i \le n$. This computation can be performed by using

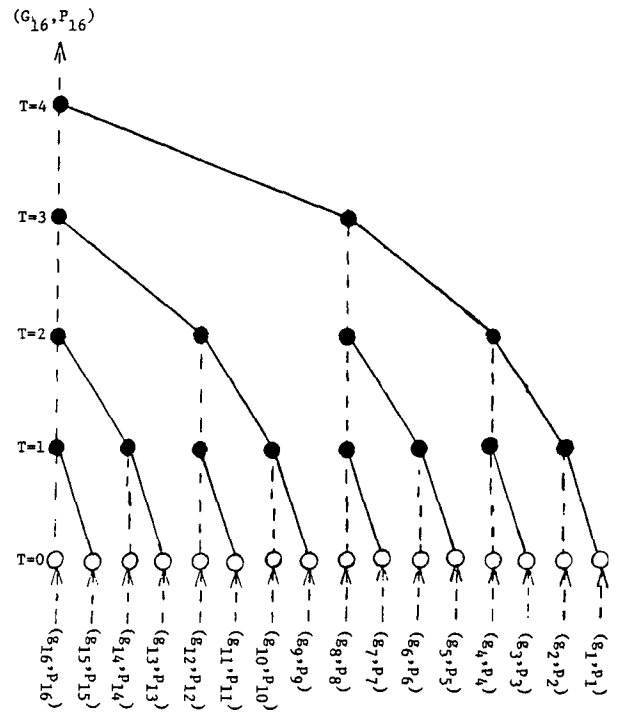

Figure 5-2: The computation of (G_{16}, P_{16}) using a tree structure.

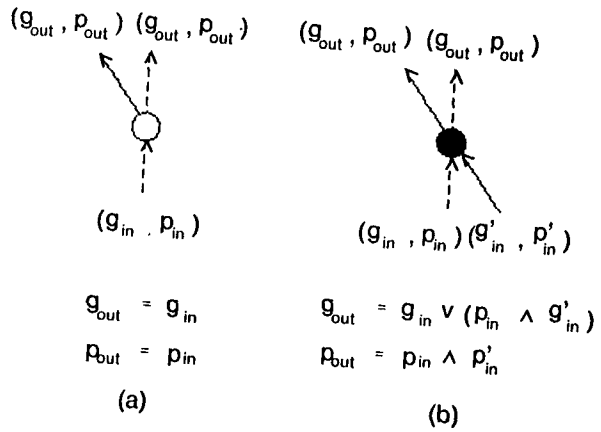

Figure 5-3: (a) The white processor, and (b) the black processor

the tree structure of Figure 5-2 once more, this time in the reverse order. We illustrate the computation, for the case n = 16, in Figure 5-4. It is easy to check that, at time T = 7, all the (G_i, P_i) are computed along the top boundary of the network. As the final outputs, we only keep the G_i which are the carries c_i.

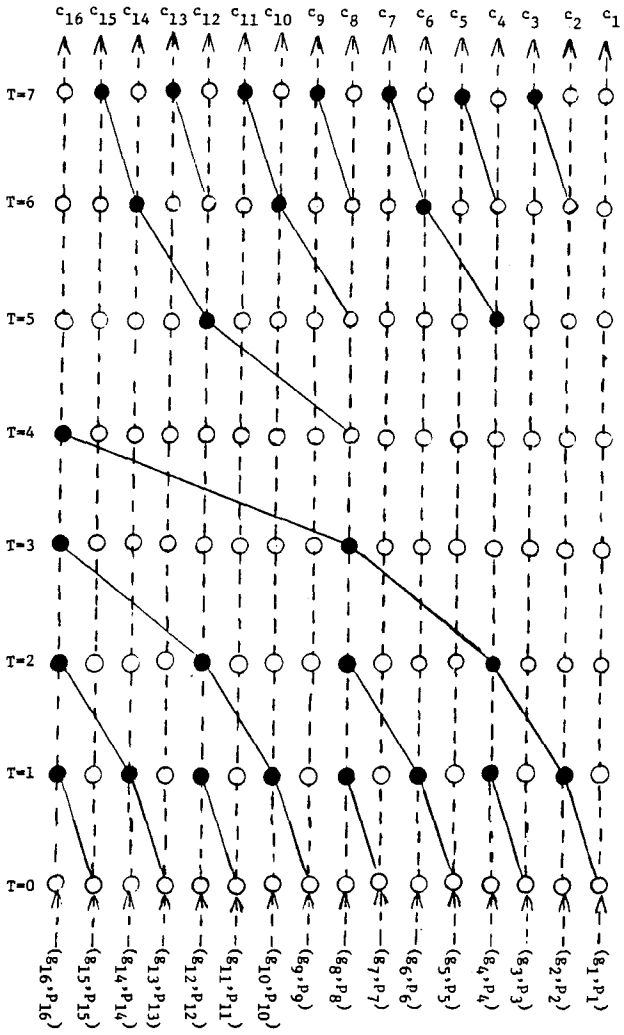

Figure 5-4: The computation of all the carries for n = 16.

In deriving the layout of Figure 5-4 we used only one distributive law. Thus, the layout could be used to evaluate arithmetic expressions of the form

$$g_n + p_n\{g_{n-1} + p_{n-1}[\ldots p_3(g_2 + p_2 g_1) \ldots]\} \qquad (5.2)$$

where g_i, p_i are numbers and the black processor in Figure 5-3 now computes $g_{out} = g_{in} + p_{in}g'_{in}$ and $p_{out} = p_{in}p'_{in}$. Note that the case $p_2 = \ldots = p_n = x$ of (5.2) is the polynomial

$$g_n + g_{n-1}x + \ldots + g_1 x^{n+1}.$$

5.3 General upper bounds for the chip complexity of binary addition

The layout shown in Figure 5-4 implies that all the carries in an n-bit addition can be computed in time O(lg n) and area O(n lg n), and therefore so can the addition itself. We show that this result is a special case of Theorem 5.1 below.

We define the width w or a parallel adder to be the number of bits it accepts at one time from each operand. For the parallel adder corresponding to the netwrok in Figure 5-4, w = 16. We have hitherto assumed that the width of a network is equal to the number n of bits in each operand. Here we consider the case w < n. We show that this case can be handled efficiently using a pipeline scheme on a network which is a modification of the one depicted in Figure 5-4.

For simplicity, assume that n is divisible by w. One can partition an n-bit integer into n/w segments, each consisting of w consecutive bits. To illustrate the idea, suppose that w = 16. Then the carry chain computation corresponding to each segment can be done on the network in Figure 5-4, and the computations for all the segments can be pipelined, starting from the least significant segment. The results coming out from the top of the network are not the final solutions, though. Results corresponding to the i^{th} least significant segment $(i > 1)$ have to be modified by applying $(G_{(i-1)w}, P_{(i-1)w})$ on the right using the operator "o." To facilitate this modification, we superimpose another tree structure on the top half of the network, as shown in Figure 5-5. Using this additional tree, the contents of the "square" processor (denoted by "□") are broadcast to all the leaves, which are black processors. The square processor, shown in Figure 5-6, is an accumulator which initially has value $(g, p) = (0, 1)$, and successively has values $(g, p) = (G_{(i-1)w}, P_{(i-1)w})$ for i = 2, 3, ... At the time when a particular $(G_{(i-1)w}, P_{(i-1)w})$ reaches the leaves, it is combined with the results just coming out from the old network there. By this pipeline scheme, we have the following result:

Theorem 5.1: Let $1 \le w \le n$. Then all the carries in an n-bit addition can be computed in time proportional to (n/w) + log w and in area proportional to w log w + 1, and so can the addition.

When w = 1, the method outlined in this section is essentially the usual serial carry-chain computation.

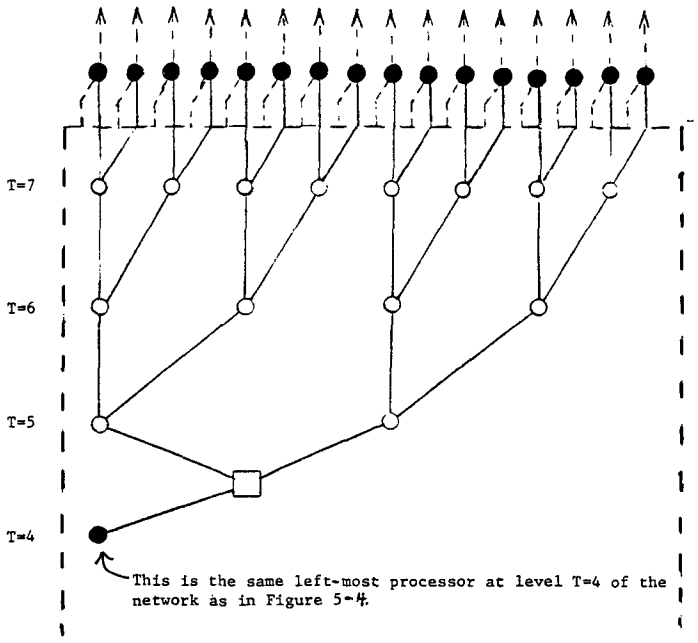

Figure 5-5: The additional tree structure to be superimposed on the top half of the network in Figure 5-4.

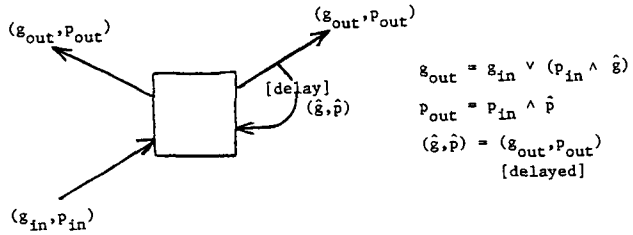

Figure 5-6: The "square" processor that accumulates $(G_{(i-1)w}, P_{(i-1)w})$.

From Theorem 5.1, we have the following

Corollary 5.1: The area-time product for n-bit addition is $O(n \lg w + w \lg^2 w + 1)$, which is $O(n \lg^2 n)$ when w = n, and $O(n \lg n)$ when $w = n/\lg n$, and O(n) when w is a constant.

One can similarly obtain an upper bound on AT^{α} for any $\alpha \ge 0$, and for each α one can choose a w to minimize the upper bound.

6. Concluding Remarks

Let $MULT_{2\alpha}(n)$ and $ADD_{2\alpha}(n)$ be the area-time complexity $AT^{2\alpha}$ for n-bit integer multiplication and addition, respectively. Note that the serial adder gives $ADD_{2\alpha}(n) = O(n^{2\alpha})$, and that for $\alpha > 1$ $MULT_{2\alpha}(n) = \Omega(n^2)$ since $A(T/\tau)^{2\alpha} > A(T/\tau)^2 \geq K_1(n/\tau)^2$. These together with Theorems 3.3 and 5.1 establish the following result:

Theorem 6.1: Under assumptions A1 to A8 of Section 2,

$$MULT_{2\alpha}(n)/ADD_{2\alpha}(n) = \begin{cases} \Omega(n^{1-\alpha}) \text{ for } 0 \leq \alpha \leq 1/2, \\ \Omega(n^{\alpha}/\lg^{2\alpha} n) \text{ for } 1/2 < \alpha \leq 1, \\ \Omega(n/\lg^{2\alpha} n) \text{ for } \alpha > 1. \end{cases}$$

Thus for any $\alpha \geq 0$, the area-time product for multiplication is asymptotically larger than that for addition. We can say that multiplication is harder than addition as far as the area-time complexity is concerned.

For binary division, it is easy to deduce a lower bound of the same form as (3.21), using the method of [Brent 76], and an upper bound $AT^{2\alpha} = O(n^{1+\alpha} \lg^{1+2\alpha} n)$, using Newton's method.

Computer arithmetic is a subject that has received intensive study in the past (see, for example, [Tung 72, Garner 76, Savage 76, Kuck 78]). Much attention has been paid to the tradeoff between time and the number of gates, but until recently little attention has been paid to the problem of connecting the gates in an economical and regular way to minimize chip area and design costs. We hope that the results of this paper should help in formalizing this new research direction of computer arithmetic, and in understanding area-time tradeoffs in the designing process.

In Section 3, we derived lower bounds on $AT^{2\alpha}$, $\alpha \in [0,1]$, for binary multiplication. Similar lower bounds on AT^2 have been obtained for computation of the discrete Fourier transform by [Thompson 79], and for that of matrix multiplication by [Savage 79]. It seems that area-time complexity is, in general, a useful measure for establishing the complexity hierarchy of many classes of problems, because it captures important attributes of a computation such as time and space, as well as communication. One should expect that more results along this line will be obtained in the near future.

References

[Abelson and Andreae 80] Abelson, H. and Andreae, P. Information Transfer and Area-Time Tradeoffs for VLSI Multiplication. *Communications of the ACM* 23:20-23, January, 1980.

[Bonneau 73] Bonneau, R.J. *A Class of Finite Computation Structures Supporting the Fast Fourier Transform.* Technical Report MAC Technical Memorandum 31, Massachusetts Institute of Technology, Project MAC, March, 1973.

[Brent and Kung 79a] Brent, R.P. and Kung, H.T. *A Regular Layout for Parallel Adders.* Technical Report, Carnegie-Mellon University, Department of Computer Science, June, 1979.

[Brent and Kung 79b] Brent, R.P. and Kung, H.T. The Area-Time Complexity of Binary Multiplication. *Journal of the ACM* (to appear), 1979. Also available as a CMU Computer Science Department technical report, July 1979.

[Brent and Kung 79c] Brent, R.P. and Kung, H.T. *On the Area of Binary Tree Layouts.* Technical Report TR-CS-79-07, The Australian National University, Department of Computer Science, July, 1979.

[Brent 70] Brent, R. P. On the addition of binary numbers. *IEEE Transactions on Computers* C-19:758-579, 1970.

[Brent 76] Brent, R.P. The Complexity of Multiple-precision Arithmetic. In Anderssen, R.S. and Brent, R.P., editors, *The Complexity of Computational Problem Solving*, pages 126-165. University of Queensland Press, Brisbane, Australia, 1976.

[Garner 76] Garner, H.L. A Survey of Some Recent Contributions to Computer Arithmetic. *IEEE Transactions on Computers* C-15:1277-1282, 1976.

[Jackson et al. 68] Jackson, L.B., Kaiser, S.F. and McDonald, H.S. An Approach to the Implementation of Digital Filters. *IEEE Trans. Audio and Electroacoust.* AU-16:413-421, September, 1968.

[Kuck 78] Kuck, D.J. *The Structure of Computers and Computations.* John Wiley & Sons, New York, 1978.

[Kung and Leiserson 79]
Kung, H.T. and Leiserson, C.E.
Systolic Arrays (for VLSI).
In Duff, I. S. and Stewart, G. W., editor, *Sparse Matrix Proceedings 1978*, pages 256-282. Society for Industrial and Applied Mathematics, 1979.
A slightly different version appears in *Introduction to VLSI Systems* by C. A. Mead and L. A. Conway, Addison-Wesley, 1980, Section 8.3.

[Ladner and Fischer 77]
Ladner, R.E. and Fischer, M.J.
Parallel Prefix Computation.
In *1977 International Conference on Parallel Processing*, pages 213-223. IEEE, 1977.

[Leiserson 80]
Leiserson, C.E.
Area-Efficient Graph Layouts (for VLSI).
Technical Report, Carnegie-Mellon University, Department of Computer Science, February, 1980.

[Linnik 44]
Linnik, U.V.
On the Least Prime in an Arithmetic Progression. I. The Basic Theorem.
Rec. Math. 15:139-178, 1944.

[Lyon 76]
Lyon, R.F.
Two's Complement Pipeline Multipliers.
IEEE Transactions on Communications COM-24(4):418-425, April, 1976.

[Mead and Conway 80]
Mead, C.A. and Conway, L.A.
Introduction to VLSI Systems.
Addison-Wesley, Reading, Massachusetts, 1980.

[Mead and Rem 79]
Mead, C.A. and Rem, M.
Cost and Performance of VLSI Computing Structures.
IEEE Journal of Solid State Circuits SC-14(2):455-462, April, 1979.

[Ofman 62]
Ofman, Y.
On the Algorithmic Complexity of Discrete Functions.
Dokl. Akad. Nauk SSSR 145:48-51, 1962.
In Russian.

[Rosser and Schoenfeld 62]
Rosser, J.B, and Schoenfeld, L.
Approximate Formulas for Some Functions of Prime Numbers.
Illinois J. Math. 6:64-94, 1962.

[Savage and Swamy 78]
Savage, J.E. and Swamy, S.
Space-Time Tradeoffs for Oblivious Sorting and Integer Multiplication.
Technical Report 37, Brown University, Department of Computer Science, 1978.

[Savage 76]
Savage, J.E.
The Complexity of Computing.
John Wiley & Sons, New York, 1976.

[Savage 79]
John E. Savage.
Area-Time Tradeoffs for Matrix Multiplication and Related Problems in VLSI Models.
Technical Report CS-50, Brown University, Department of Computer Science, August, 1979.

[Schonhage and Strassen 71]
Schonhage A. and Strassen, V.
Schnelle Multiplikation grosser Zahlen.
Computing 7:281-292, 1971.

[Thompson 79]
Thompson, C.D.
Area-Time Complexity for VLSI.
In *Proc. Eleventh Annual ACM Symposium on Theory of Computing*, pages 81-88. ACM, May, 1979.

[Tung 72]
Tung, C.
Arithmetic.
In Cardenas, A.F., Press, L. and Marin, M.A., editors, *Computer Science.* Wiley-Interscience, New York, 1972.

[Wagstaff 79]
Wagstaff, S.S., Jr.
Greatest of the Least Primes in Arithmetic Progressions Having a Given Modulus.
Math. Comp. 33:1073-1083, 1979.

[Wallace 64]
Wallace, C.S.
A Suggestion for a Fast Multiplier.
IEEE Trans. Elec. Comp. EC-13:14-17, 1964.

[Winograd 65]
Winograd, S.
On the Time Required to Perform Addition.
Journal of the ACM 12(2):277-285, 1965.

[Yaglom and Boltyanskii 61]
Yaglom, I.M. and Boltyanskii, V.G.
Convex Figures.
Holt, Rinehart and Winston, New York, 1961.
Translated by P.J. Kelly and L.F. Walton.