Towards Concurrent Arithmetic: Residue Arithmetic and VLSI

Chao-Lin Chiang

Computer Science Department
California Institute of Technology

5135:TR:84

Towards Concurrent Arithmetic: Residue Arithmetic and VLSI

by

Chao-Lin Chiang

5135:TR:84

Department of Computer Science

California Institute of Technology

Pasadena, California

Submitted to the Computer Science Department in Partial Fulfillment of the Requirements for the Degree of Master of Science.

The research was sponsored in part by the Defense Advanced Research Projects Agency (DARPA), and monitored by the Office of Naval Research (ONR) under contract N00014-79-C-0597.
The research was also in part sponsored by the Office of Naval Research under contract N00014-84-K-0043.

Acknowledgments

Special thanks are due to my advisor, Lennart Johnsson, for his continued guidance and support on this research.

I also wish to thank Tzu-mu Lin for his help and encouragement.

Many thanks to everyone in the computer science department for their willingness to help and to be friendly.

Table of Contents

Part 2. Distributed RC Delay Line Model and MOS PLA Timing Estimation

Introduction

Arithmetic based on the residue number system is attractive because addition and multiplication are performed on each digit independently [Ga59]. It is a "concurrent" arithmetic. There are many important issues in residue arithmetic. Two such issues are the degree of concurrency and the code efficiency as functions of range requirements. From an implementation standpoint, it is important to find a good implementation of residue adders and multipliers for different ranges, and to select a basis for a residue number system satisfying the specific range requirement. Choosing a configuration that fully explores the potential of the residue number system at various stages of the technology is one of the major concerns of this paper.

There are many alternatives for the implementation of a residue multiplier. Different designs are often evaluated in terms of two factors: cost and speed. Traditionally, the cost is evaluated by the number of IC components used. In VLSI, silicon area is more relevant. The signal delay in VLSI circuits is no longer determined by the number of gates in a path. Wire delays need to be included in realistic delay models. In [Ch83a], a model is developed to estimate the delay of MOS storage structures such as PLA and ROM. The details of this model is contained in the second part of this report.

This report is organized as follows: In part 1, the residue number system is introduced first. Then, several different designs of a multiplier based on the residue number system are presented and analyzed. Finally, base selection algorithms with respect to minimum delay or minimum area are presented. In part 2, "Distributed RC Delay Line Model and MOS PLA Timing Estimation", a VLSI computation model for storage structures is developed in detail.

Part 1. Residue Arithmetic and VLSI

1. Introduction

1.1 Weighted Number Systems

The decimal number system and the binary number system are ***weighted*** number systems. A number system is said to be weighted if there exists a set of weights w_i's such that, for any number x, x can be expressed in the form $x = \sum_i a_i w_i$, where the a_i's are a set of permissible digits. If the values for w_i's are successive powers of the same number, then the number system has a fixed ***base*** or fixed ***radix***, e.g., base 10 for the decimal number system, base 2 for the binary number system.

The fixed base number system has the following important advantages:

- Relative-magnitude comparison of two numbers can be mechanized easily.
- The logic required for performing arithmetic is essentially identical for all digits.
- Overflow detection is easily mechanized.

The attributes which lead to these advantages impose a limitation on the speed with which many arithmetic operations can be performed. In both the binary and the decimal number system, truly parallel arithmetic operations, i.e., processing all digits concurrently, is not possible because of carry propagation. For all arithmetic operations in such number systems, each digit of the result is a function of all digits of equal or lower significance. This characteristic imposes a fundamental limitation on the speed of arithmetic in weighted number systems.

In order to achieve high-speed computation, other number system that allows concurrent arithmetic is needed. Such a number system is the residue number system.

1.2 The Residue Number System

In the residue number system [Ga59], a number is represented by several residue digits. The arithmetic operations addition, subtraction and multiplication are inherently carry-free, i.e., each digit of the result is a function of only one digit from each of the operands. In particular, for the case of multiplication, the need for partial products is eliminated. This local property allows a high degree of concurrency.

The residue number system is not a weighted number system, however, and does not have many of the advantages listed above for such systems. Operations such as division, magnitude comparison and overflow detection are difficult in a residue number system. Several attempts have been made to find efficient algorithms to solve these problems [Ba69], [Ki73], [Ba80]. However, due to the inherent non-local properties in these operations [Sz67], no efficient algorithms with complexity comparable to addition and multiplication have been found. The complexity of these operations has prevented the residue number system from use in general purpose computers.

However, there is a large class of digital signal processing and pattern recognition problems in which addition and multiplication dominates [Je77], [Je79], [Ts79], [Hu81], [Fo82]. In those applications, the residue arithmetic is competitive with binary arithmetic. In [Je77], a residue number system is used to implement a 64th order bandpass FIR filter. In [Ts79], a hardware implementation of a FFT based on the residue number system is proposed. In [Fo82], a VLSI architecture for pattern recognition using residue arithmetic is described. Other applications can be found in [Je79] and [Hu81]. In these applications, the required range is typically small (10 to 24 bits) and the individual residue adders and multipliers are implemented by table look-up. The size of the tables ranges from 1K to 5K bits and commercial high-speed ROM's are used to realize the tables. In the references above, the cost is measured in terms of IC components used and the performance comparisons are based on particular cases.

In the era of VLSI technology, hundreds of thousands of devices can be integrated on one chip. Therefore, the need for a new criteria to evaluate different VLSI designs is evident. Area is a measurement that corresponds to counting IC components.

In residue arithmetic, the table look-up scheme is one design alternative of many. Two factors make the table look-up scheme unfavorable. First, the table size grows exponentially with the number of inputs. For a large range requirement, the table size will soon become too large. Second, the MOS PLA or ROM that implements a table contains long poly wires. The wire delay does not scale well as feature sizes scale down. Hierarchically organized tables or storages as described in [Me80] can be used to reduce the delay to logarithmic in the table size, instead of proportional to its size. In this report, the effect of these two factors - increased range requirement and reduced feature size - on different design alternatives is analyzed in detail.

In the next section, the Chinese Remainder Theorem and some basic concepts of a residue number system are introduced. Two important issues, the "degree of concurrency" and the "code efficiency," are discussed. In section 3, several implementations of residue adders and multipliers are discussed. Based on

VLSI computation models, the cost functions with respect to time and area are calculated and the relative competent range of each implementation is analyzed in section 4. In section 5, optimal base selection algorithms with respect to time and area are presented.

2. Residue Arithmetic and the Chinese Remainder Theorem

2.1 Congruence relations, Modulo operations and the Chinese Remainder Problem

Consider three integers A, α and b. If there exists another integer t such that $A = \alpha + b\,t$, then A, α and b satisfy the ***congruence relation***

$$A \equiv \alpha \pmod b$$

which is read, A is congruent to α modulo b. α is called the ***residue*** and b is the ***modulus*** of the number A. For convenience, $A \pmod b$ is sometimes abbreviated as $|A|_b$.

The problem of computing solutions to systems of simultaneous congruence relations, such as

$$\begin{aligned} x &\equiv 1 \bmod 5 \\ x &\equiv 0 \bmod 7 \\ x &\equiv 4 \bmod 8 \end{aligned}$$

was studied by Chinese mathematicians as far back as the first century A.D. In deference to this historical background, we call the problem of solving a system of n congruence relations

$$x \equiv r_k \bmod m_k \qquad (k = 0, \ldots, n-1)$$

the Chinese Remainder Problem.

It is readily verified that $x = 196$ is a solution to the above problem. The general algorithm for computing solutions to Chinese Remainder Problems is based on the Chinese Remainder Theorem [Sz67], [Ah74].

2.2 Concurrent Computations in the Residue Number System

Let $\{m_i \mid \gcd(m_i, m_j) = 1, \text{ for } i \neq j, \quad i,j = 1,..,p\}$ be a set of pairwise relatively prime numbers and R $= \prod_{i=1}^{p} m_i$ be the *range* of the set. According to the Chinese Remainder Theorem, any number X in the range [0, R-1] can be represented uniquely by a set of residues $(x_1, ..., x_p)$, where $x_i \equiv X \bmod m_i$ is called the ***residue digit*** with respect to ***modulus*** m_i and the p-tuple $(x_1, ..., x_p)$ is a ***residue code*** of the number X. The set $\{m_1, \ldots, m_p\}$ is called the ***base*** of the residue number system. Table 1 shows the residue number representation corresponding to the integers 0 to 29 with respect to moduli 2, 3, 5.

A number represented in residue code can be converted to binary code according to the formula [Sz67] :

$$X \equiv \prod_{i=1}^{p} x_j \cdot \hat{m}_j \cdot \left| \frac{1}{\hat{m}_j} \right| \quad \bmod R$$

where $\quad \hat{m}_j = \dfrac{R}{m_j} \quad$ and $\quad \left| \dfrac{1}{\hat{m}_j} \right| = a \leftrightarrow (\hat{m}_j \cdot a) \equiv 1 \bmod R$

Integers	Residue Digits			Integers	Residue Digits		
	Moduli				Moduli		
	2	3	5		2	3	5
0	0	0	0	15	1	0	0
1	1	1	1	16	0	1	1
2	0	2	2	17	1	2	2
3	1	0	3	18	0	0	3
4	0	1	4	19	1	1	4
5	1	2	0	20	0	2	0
6	0	0	1	21	1	0	1
7	1	1	2	22	0	1	2
8	0	2	3	23	1	2	3
9	1	0	4	24	0	0	4
10	0	1	0	25	1	1	0
11	1	2	1	26	0	2	1
13	0	0	2	27	1	0	2
13	1	1	3	28	0	1	3
14	0	2	4	29	1	2	4

Table 1. Residue representation of the numbers 0 to 29 for moduli 2, 3, 5

Example: Conversion of a residue number by use of the Chinese Remainder Theorem.

For moduli $m_1 = 13$, $m_2 = 11$, $m_3 = 7$, and $m_4 = 9$, find the number whose residue representation is $\{4, 2, 4, 7\}$.

Solution

$$R = \prod_{i=1}^{4} m_i = 13 \times 11 \times 7 \times 9 = 9009$$

$$|\hat{m}_1|_{m_1} = |11 \times 7 \times 9|_{13} = |693|_{13} = 4 \quad \text{then} \left|\frac{1}{\hat{m}_1}\right|_{13} = 10$$

$$|\hat{m}_2|_{m_2} = |13 \times 7 \times 9|_{11} = |819|_{11} = 5 \quad \text{then} \left|\frac{1}{\hat{m}_2}\right|_{11} = 9$$

$$|\hat{m}_3|_{m_3} = |13 \times 11 \times 9|_{7} = |1287|_{7} = 6 \quad \text{then} \left|\frac{1}{\hat{m}_3}\right|_{7} = 6$$

$$|\hat{m}_4|_{m_4} = |13 \times 11 \times 7|_{9} = |1001|_{9} = 2 \quad \text{then} \left|\frac{1}{\hat{m}_4}\right|_{9} = 5$$

$$\begin{aligned} |x|_{9009} &= \big|693 \times |10 \times 4|_{13} + 819 \times |9 \times 2|_{11} + 1287 \times |6 \times 4|_7 + 1001 \times |5 \times 7|_9\big|_{9009} \\ &= |693 \times 1 + 819 \times 7 + 1287 \times 3 + 1001 \times 8|_{9009} \\ &= |18295|_{9009} = 277 \end{aligned}$$

Let X and Y have residue codes $(x_1, ..., x_p)$ and $(y_1, ..., y_p)$ respectively and $X, Y, X+Y, X \cdot Y \in [0, \text{R-1}]$. Then [Ga59]

$$|X+Y|_{m_i} = |x_i + y_i|_{m_i} \quad \text{and} \quad |X \cdot Y|_{m_i} = |x_i \cdot y_i|_{m_i}$$

It follows that

$$(|X+Y|_{m_1}, ..., |X+Y|_{m_p}) = (|x_1 + y_1|_{m_1}, ..., |x_p + y_p|_{m_p})$$

$$(|X \cdot Y|_{m_1}, ..., |X \cdot Y|_{m_p}) = (|x_1 \cdot y_1|_{m_1}, ..., |y_p \cdot y_p|_{m_p})$$

Example: Addition in the residue number system

For the moduli 4,3,5 and 11 add $x = 102 \leftrightarrow \{2, 0, 2, 3\}$ and $y = 211 \leftrightarrow \{3, 1, 1, 2\}$.

Solution:

$$\begin{array}{rccccc} & \text{Moduli:} & 4 & 3 & 5 & 11 \\ \hline 102 \longleftrightarrow & & \{2, & 0, & 2, & 3\} \\ +\underline{211} \longleftrightarrow & & \{3, & 1, & 1, & 2\} \\ \hline |313|_{660} = 313 \longleftrightarrow & & \{1, & 1, & 3, & 5\} \end{array}$$

Example: Multiplication in the residue number system

For the moduli 4,3,5 and 11 multiply $x = 25 \leftrightarrow \{1, 1, 0, 3\}$ and $y = 21 \leftrightarrow \{1, 10, 1, 10\}$.

Solution:

$$\begin{array}{rccccc} & \text{Moduli:} & 4 & 3 & 5 & 11 \\ \hline 25 \longleftrightarrow & & \{1, & 1, & 0, & 3\} \\ \times\underline{21} \longleftrightarrow & & \{1, & 0, & 1, & 10\} \\ \hline |525|_{660} = 525 \longleftrightarrow & & \{1, & 0, & 0, & 8\} \end{array}$$

The examples above show that addition and multiplication of two numbers can be carried out on each digit independently. When these operations are performed in each residue digit concurrently, the overall speed is often limited by the residue digit with respect to the largest modulus. For a given range R, if the moduli are properly chosen such that the maximum modulus is as small as possible then the maximum speed-up can be achieved. However, the maximum modulus for a given range can not be arbitrarily small since each pair of moduli must be relatively prime numbers. Indeed, there is an analytic relation between the required range and the maximum modulus. This relation indicates the inherent degree of concurrency in the residue number system.

2.3 Degree of Concurrency and Code Efficiency in the Residue Number System

Consider a residue number system with base $\{m_1, \ldots, m_p\}$ and range R $= \prod_{i=1}^{p} m_i$. Each residue digit is typically encoded in the binary number system. Let M_i be the number of bits required to encode the i-th residue digit and N_b be the number of bits required for a direct binary encoding of the largest number in the range [0, R-1], i.e., $M_i = \lceil \log m_i \rceil$, $N_b = \lceil \log R \rceil$, where $\log = \log_2$.

Denote $M = \max\{M_i\}$ the number of bits required to encode the largest moduli and $N_r = \sum_1^p M_i$ the total number of bits required to encode the entire residue code. The quantity $\gamma = N_b/M$ is the ratio between the number of bits needed in the binary number system and the number of bits for the largest modulus in a binary encoded residue number system. We define it as the ***degree of concurrency*** of the residue number system. Another quantity $\eta = N_b/N_r$ is the ratio of the total number of bits required to encode numbers with equal range in a binary number system and a residue system. We define it as the ***code efficiency*** of the residue number system.

Table 2 lists several bases for some ranges and the accompanying values of M, N_b , N_r , γ and η. Note that all moduli are of the form p^k, where p is prime. Moduli are listed in order of increasing p. The reason for choosing these particular bases will be clear later.

From table 2, it is easily seen that the degree of concurrency (γ) is monotoni-

moduli	M	N_b	N_r	γ	$\log N_b/M$	η
8,3,5,7	3	10	12	3.33	1.11	0.91
16,9,5,7,11,13	4	20	22	5.00	1.08	0.91
32,27,25,7,11,13,17,19,23,29,31	5	48	51	9.60	1.12	0.94
64,27,25,49,11,13, ... ,53,59,61	6	90	97	15.00	1.08	0.93
128,81,125,49, ... ,107,109,113	7	184	196	26.29	1.08	0.94
256,243,125,49, ... ,239,241,251	8	363	386	45.37	1.06	0.94

Table 2. Degree of concurrency (γ) and code efficiency (η)

cally increasing with the range. Indeed $(\log N_b)/M$ approaches 1 as the range increases. This follows from the fundamental theorem of arithmetic. A proof is given below. The significance of this property is that instead of N_b-bit arithmetic, $\log N_b$-bit binary arithmetic suffices. However, $O(N_b/\log N_b)$ such units are required instead of one N_b-bit unit.

Since each residue digit is binary encoded, only m_i out of 2^{M_i} available codes are used. There is some redundancy in the residue code. Therefore the code efficiency is always less than 1. However, table 2 shows that the code efficiency is greater than 0.9 for ranges of interest and it can further be proved that $N_b/N_r \to 1$ as $N_b \to \infty$.

That the property $O(M)=O(\log N_b)$ follows from the fundamental theorem of arithmetic will now be shown.

Definition 1. $\{m_1,\ldots,m_p\}$ is a *feasible base* for a range R if any two elements in $\{m_1,\ldots,m_p\}$ are relatively prime and the product of all elements in $\{m_1,\ldots,m_p\}$ is greater or equal to R, i.e., $\gcd(m_i,m_j) = 1$ for $i \neq j$, $i,j = 1,..,p$ and $\prod_{i=1}^{p} m_i \geq R$. Denote the largest modulus in $\{m_1,\ldots,m_p\}$ as the *norm* of the base.

For example, $\{3,5,7,11\}$ is a feasible base for the range R=1000, but not a feasible base for the range R=1200, since $3 \times 5 \times 7 \times 11 = 1155 < 1200$. The norm of this base is 11. $\{6,7,9,11\}$ is not a feasible base for any range since 6 and 9 are not relative primes.

Definition 2. Let x be the norm of a feasible base $\{m_1, \ldots, m_p\}$. Then this base is also an *optimal base* if for any other set $\{n_1, \ldots, n_q \mid \gcd(n_i, n_j) = 1, \text{ for } i \neq j, \ i,j = 1,..,q, \text{ and } n_j \leq x, \forall j\}$, $\prod_{i=1}^{q} n_j \leq \prod_{i=1}^{p} m_i$.

Definition 2 means that if a base is constructed with the restriction that its norm is less than x, then the range has an upper bound R, which is achieved by the optimal base. The upper bound R is a function of x, i.e., R=$\mathcal{R}(x)$. The following lemma gives the construction of an optimal base and the characteristic of $\mathcal{R}(x)$.

Lemma 1. (Construction of an Optimal Base) For any $x \geq 2$, the set of moduli $\{p_i^{k_i} \mid p_i : \ i\text{th prime}, \ p_i^{k_i} \leq x < p_i^{k_i+1}\}$ forms an optimal base with range R= $\mathcal{R}(x) = \prod_{\forall p_i^{k_i} \leq x} p_i^{k_i}$ where $k_i = \left\lfloor \frac{\log x}{\log p_i} \right\rfloor$.

Proof:

(1) $\{p_i^{k_i}\}$ is a feasible base since $p_i^{k_i}$'s are relatively prime for different i

(2) Let $\{m_1, \ldots, m_p\}$ be the given base and $\{n_1, \ldots, n_q\}$ be another feasible base. By the fundamental theorem of arithmetic [Ha79], there exist a unique *prime expansion* for the range of the base $\{m_1, \ldots, m_p\}$, and for the range of the base $\{n_1, \ldots, n_q\}$.

$$\prod_{i=1}^{p} m_i = \prod_{i=1}^{r} p_i^{k_i} \tag{1}$$

and

$$\prod_{j=1}^{q} n_j = \prod_{i=1}^{s} p_i^{l_i} \qquad (2)$$

where p_i is the ith prime number.

Assume $n_j \leq \max\{m_i\}$ for all $1 \leq j \leq q$, then for every term $p_i^{l_i} \neq 1$ in eq(2), $p_i^{l_i} \leq$ some $n_j \leq \max\{m_j\} \leq x$. The first $\leq$ holds because each $p_i^{l_i}$ divides exactly one n_j (otherwise, elements in $\{n_1, \ldots, n_q\}$ are not relatively prime.) The second and third $\leq$ hold due to the assumptions. Compare eq(1) with eq(2) term by term. In eq(2) $p_i^{l_i} \leq x$, while in eq(1) $p_i^{k_i} \leq x < p_i^{k_i+1}$, hence $l_i \leq k_i$ for all i. Therefore $\prod_{j=1}^{p} n_j \leq \prod_{i=1}^{q} m_i$.

Lemma 2. $\log(\mathcal{R}(x)) = \log\left(\prod_{\forall p_i^{k_i} \leq x} p_i^{\lfloor \frac{\log x}{\log p_i} \rfloor}\right)$ is of order x.

Proof: see [Ha79] pp. 341.

Theorem Let N_b be the number of bits required for a direct binary encoding of numbers in the range [0, R-1] and M be the number of bits required for binary encoding of the maximum possible digit in a residue representation of the same numbers. Then $M=O(\log N_b)$ provided that the base is an optimal base with range $\geq$ R.

Proof:

Let $\mathcal{M} = \mu x[\mathcal{R}(x) \geq \mathrm{R}]$ where μ is the minimalization operator. Then the optimal base with norm $\mathcal{M}$ achieves range $\geq$ R. By the definition of $\mathcal{M}$, $\mathcal{R}(\mathcal{M}-1) < \mathrm{R} \leq \mathcal{R}(\mathcal{M})$, i.e., $\log \mathcal{R}(\mathcal{M}-1) < \log \mathrm{R} \leq \log \mathcal{R}(\mathcal{M})$. By Lemmas 1 and 2, $O\big(\mathcal{M}\big)=O\big(\log \mathrm{R}\big)$. Since $\mathrm{M}=\lceil \log \mathcal{M} \rceil$ and $\mathrm{N_b} = \lceil \log \mathrm{R} \rceil$, it follows that $\mathrm{M}=O\big(\log \mathrm{N_b}\big)$.

3. Implementations of Residue Arithmetic

The concurrent property of the residue number system provides a method to achieve high-speed computation. However, addition and multiplication of each digit is more complex than in the binary number system in that it is performed modulo the set of integers, $m_1, .., m_p$. For moduli of the form 2^n, $2^n + 1$ or $2^n - 1$, the modulo operation is essentially for free and implementations of residue arithmetic is simple [Et82], [Ta82]. However, if the selection of moduli is limited to the form $2^n, 2^n + 1$ or $2^n - 1$, ("easy" base) the degree of concurrency will be reduced since the available choices of moduli are fewer and the maximum modulus is larger, see Table 3. In fact the code efficiency is slightly higher for the easy base, but the number of bits needed for the maximum base rapidly becomes significantly larger.

Performing the modulo operation by a restoring or nonrestoring division algorithm is very time consuming and should be avoided. Finding a good algorithm for the modulo operations therefore becomes an essential part of realizing an efficient residue arithmetic system. We will now discuss a few alternatives for the implementation of multiplication modulo arbitrary (but fixed) integers.

Addition and multiplication modulo the integers $m_1, m_2, .., m_p$ are traditionally implemented by table look-up [Je77], [Hu81], [Ba82], [Fo82]. The table look-up scheme provides many advantages: it is programmable and easy to design; it has a very regular structure and can be made very dense. Since the table size grows exponentially with the number of inputs, it is almost impossible to implement a 10-bit by 10-bit adder or multiplier in the binary number system ($2^{20} \approx$1M bits!). However it is possible to decompose a large number, say 18 bits, into several 3 or 4-bit residue digits (see Table 2) so that the table size can be kept small when implementing residue adders or multipliers in each residue digit.

optimal base						easy base					
N_b	Norm	M	N_r	γ	η	N_b	Norm	M	N_r	γ	η
10	8	3	12	3.33	0.91	9	8	3	11	3.00	0.81
20	16	4	22	5.00	0.91	20	32	5	22	4.00	0.91
48	32	5	51	9.60	0.94	54	1024	10	56	5.40	0.96
90	64	6	97	15.50	0.93	95	16384	15	96	6.33	0.99
184	128	7	196	26.29	0.94	180	4194304	22	181	8.18	0.99

Table 3. The norms, degree of concurrency, and code efficiency for an optimal base and an easy base

With residue encoding of the operand, they can be decomposed into $O(N_b/\log N_b)$ residue digits. Each digit is encoded by at most $O(\log N_b)$ bits. If the table look-up scheme is used to implement a two-operand adder or multiplier in the binary number system, the table size is $O(N_b \cdot 2^{2N_b}) = O(N_b \cdot 4^{N_b})$ bits. On the other hand, if the adder or multiplier is implemented for residue encoded operands, then each table is of size $O(\log N_b \cdot 2^{2\log N_b}) = O(\log N_b \cdot N_b{}^2)$ bits and the total size is $O((N_b/\log N_b) \cdot \log N_b \cdot N_b{}^2) = O(N_b{}^3)$. The advantage of the residue number system is clear since the table size in the binary number system grows exponentially, while in the residue number system it grows at a polynomial (cubic) rate.

The table look-up scheme without partitioning has some drawbacks, however. First, $O(N_b{}^3)$ is still too big for a large range requirement. Second, a large storage contains long wires unless hierarchically organized. The wire delay does not decrease with reduced feature size and eventually becomes a performance limiting factor. A hierarchical storage as described in [Me80] has an access time that grows logarithmically with the storage size. A few alternatives to the table look-up method will be discussed in the following subsections. The time-space complexity of each design is analyzed. For the time complexity of storage structures the model derived in the second half of this report is used.

3.1 A Residue Adder

Residue addition $z_i = (x_i + y_i) \bmod m_i$, where $0 \le x_i, y_i \le m_i - 1$, can be computed as

$$z_i = \begin{cases} x_i + y_i & \text{if } x_i + y_i < m_i \\ x_i + y_i - m_i & \text{if } x_i + y_i \ge m_i \end{cases}$$

One M_i-bit adder can be used to compute $x_i + y_i$ while another computes $x_i + y_i - m_i$. The carry bit generated from the second adder indicates whether or not $x_i + y_i$ is greater than m_i. A multiplexor controlled by the carry bit selects the correct output, see Figure 1. In this implementation, $2M_i$ 1-bit adders and M_i multiplexors are needed.

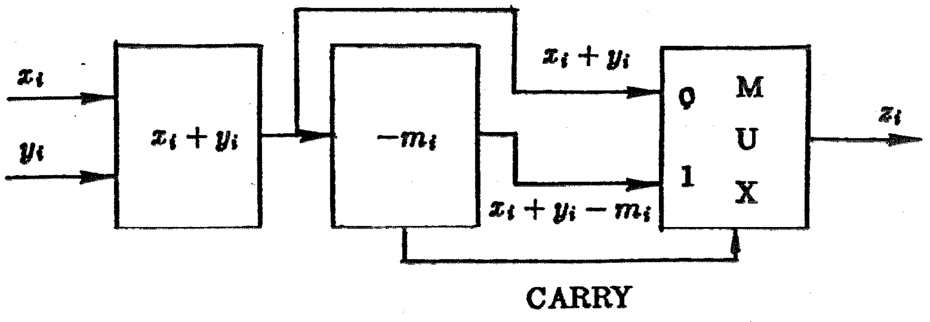


Figure 1. An implementation of residue addition

3.2 Residue Multipliers

Three different methods to implement multipliers for one residue digit have been evaluated:

1. pure table look-up
2. combinations of tables and adders
3. arrays of adders

3.2.1 Pure Table Look-up

A table for a one-digit residue multiplier has two M_i-bit inputs, x_i and y_i, and produces one M_i-bit output, $z_i = (x_i \cdot y_i) \bmod m_i$, see Figure 2. The total number of bits in one table is $M_i 2^{2M_i} = M_i 4^{M_i}$. For $O(N_b / \log N_b)$ digits, the total table size is $\sum M_i 4^{M_i} \approx N_b^3$ bits.

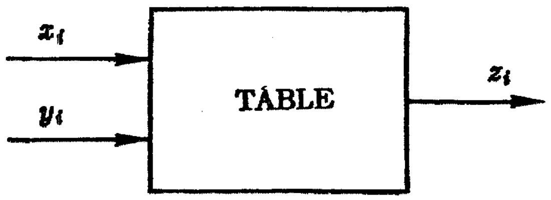


Figure 2. Table look-up residue multiplier

3.2.2.a A Quarter Square Residue Multiplier [Da65]

Figure 3 shows another type of residue multiplier. It realizes the identity:

$$|x_i \cdot y_i|_{m_i} = \left| \left| \frac{(x_i + y_i)^2}{4} \right|_{m_i} - \left| \frac{(x_i - y_i)^2}{4} \right|_{m_i} \right|_{m_i}$$

The "modular quarter square" $\left|\frac{(\cdot)^2}{4}\right|_{m_i}$ is implemented by table look-up. There are two tables for each residue digit. Each table has $M_i 2^{M_i+1} = 2M_i 2^{M_i}$ bits. For $O(N_b/\log N_b)$ digits, the total table size $= \sum 2 \cdot 2M_i 2^{M_i} \approx (N_b/\log N_b) \cdot (4\log N_b) \cdot 2^{\log N_b} = 2N_b^2$ bits. Note that the table size is $O(N_b{}^2)$ compared to $O(N_b{}^3)$ if a single table is used. On the other hand, one M_i-bit binary adder/subtractor and one M_i-bit residue subtractor are needed for each residue digit. The total number of 1-bit adder/subtractors is of order $O(N_r)$.

3.2.2.b An Index Transform Residue Multiplier

One familiar method of multiplication is " log transform – addition – antilog transform " [Ki71]. Similarly, residue multiplication can be performed by " index transform – residue addition – reverse index transform ". The index transform IND_r is defined by [Sz67]:

$$IND_r C \equiv b \bmod P \quad \leftrightarrow \quad r^b \equiv C \bmod P, \quad P \text{ prime}$$

For m_i a prime number, an implementation of a residue multiplier based on the equation :

$$|x_i \cdot y_i|_{m_i} = \left| r^{(IND_r x_i + IND_r y_i) \bmod (m_i - 1)} \right|_{m_i}$$

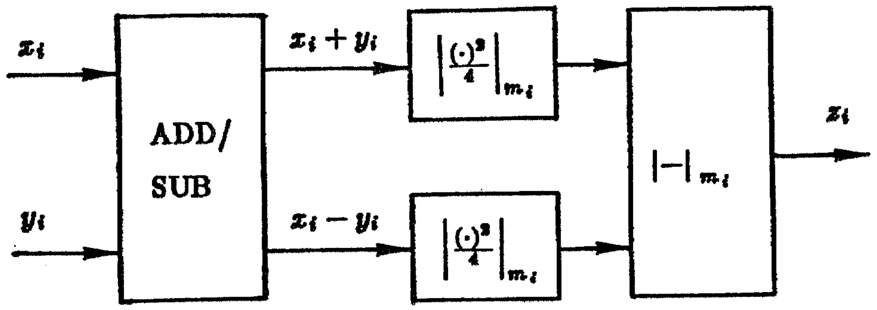

x_i
y_i
ADD/ SUB
$x_i + y_i$
$x_i - y_i$
$\left|\frac{(\cdot)^2}{4}\right|_{m_i}$
$\left|\frac{(\cdot)^2}{4}\right|_{m_i}$
$|-|_{m_i}$
z_i

Figure 3. A quarter square residue multiplier

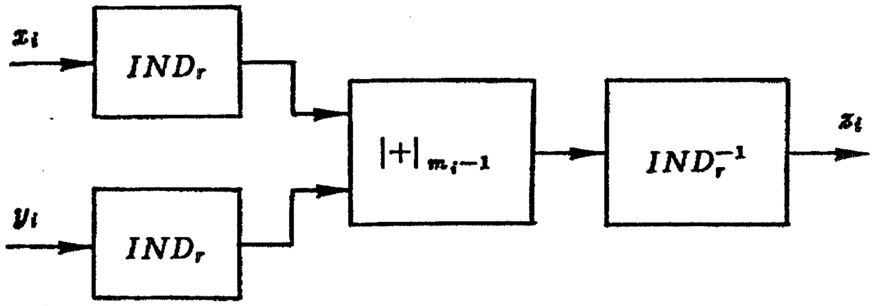


Figure 4. An index transform residue multiplier

is shown in Figure 4.

The index and reverse index transform is implemented by tables. For each residue digit, there are two tables for the index transform and one for the reverse index transform. In each table, both the input and the output have M_i bits. The table size is $3M_i 2^{M_i}$ bits. For $O(N_b/\log N_b)$ digits, the total table size is $\approx (N_b/\log N_b) \cdot 3 \log N_b 2^{\log N_b} = 3N_b^2$. There are M_i 1-bit adder/subtractors in one digit. The total number of 1-bit adders is $O(N_r)$.

3.2.3 An Array Multiplier

A residue array multiplier is composed of two parts: (1) a binary array multiplier and (2) a converter performing the modulo evaluation of the product. The binary multiplier has a delay of $O(M_i)$ and an area of $O(M_i^2)$. The complexity of the conversion part can be made equal to that of the binary multiplier by employing carry save adders [Hw79].

Let A be the $2M_i$ bit product, m_i be the M_i bit modulus, and z_i the M_i bit output number, $z_i = A \bmod m_i$. To compute z_i, the distributive property of the modulo operation over addition is used:

$$\left(a_{2M_i-1} 2^{2M_i-1} + \ldots + a_{M_i} 2^{M_i} + a_{M_i-1} 2^{M_i-1} + \ldots a_0 2^0\right) \bmod m_i =$$
$$= \left(a_{2M_i-1} \underline{(2^{2M_i-1} \bmod m_i)} + \ldots + a_{M_i} \underline{(2^{M_i} \bmod m_i)} + \underline{(a_{M_i-1} 2^{M_i-1} + \ldots + a_0 2^0)} \right)$$

The M_i highest order bits a_{2M_i-1}, a_{2M_i-2}, ..., a_{2M_i} can be used to control whether or not the M_i-bit constants $(2^{2M_i-1} \bmod m_i)$, $(2^{2M_i-2} \bmod m_i)$, ..., $(2^{2M_i} \bmod m_i)$ shall be added to the M_i lowest order bits, Figure 5.

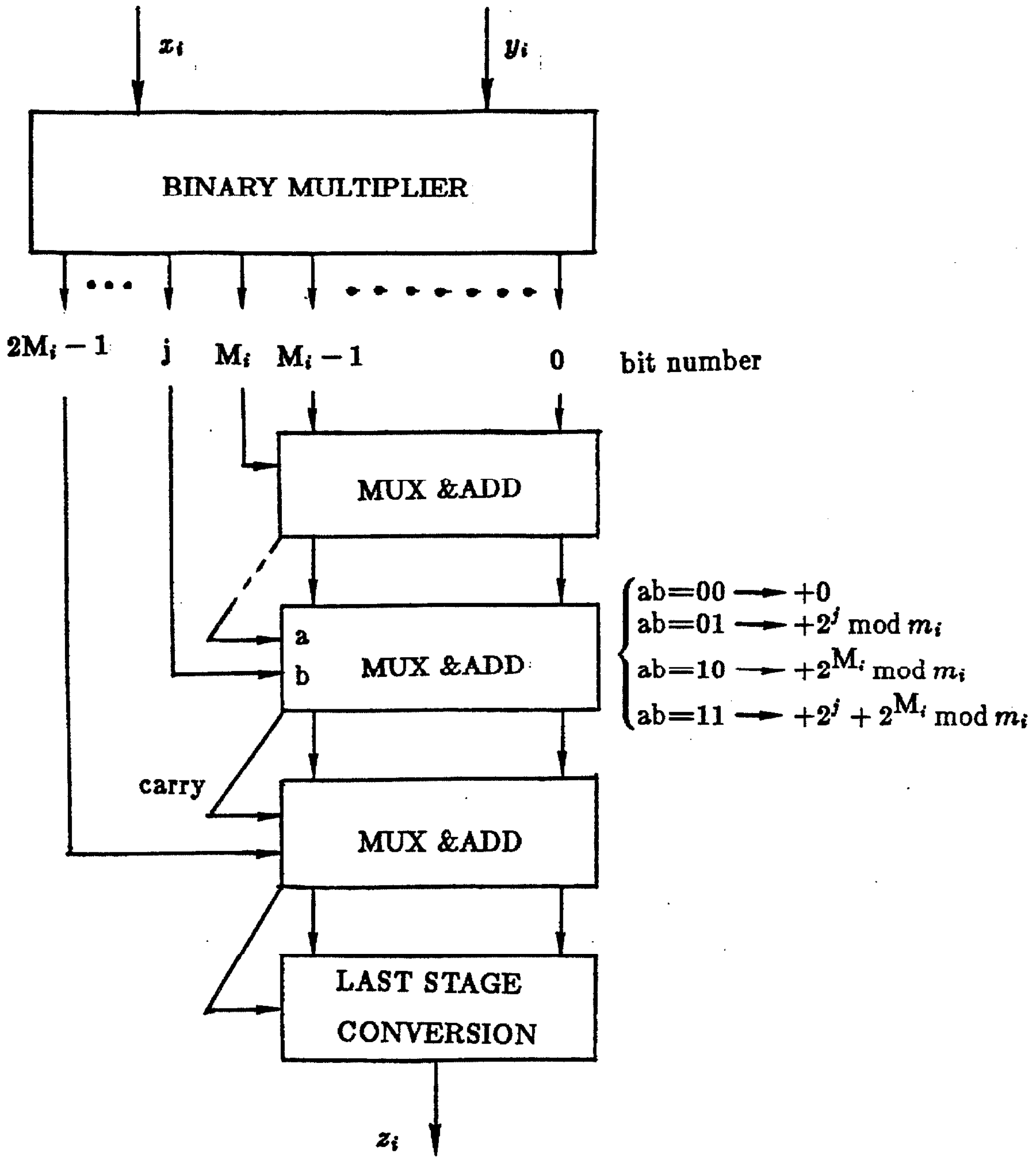
The conversion part consists of M_i stages of M_i-bit adders. Each adder stage needs $O(M_i)$ time with the carry propagating from one bit to the next. The total delay is $O(M_i^2)$. However, if the carry for each bit is fed into the next stage only $O(M_i)$ time is needed. An additional stage is needed to convert $M_i + 2$ bit numbers into M_i bit residue numbers.

For one residue array multiplier, the area is $O(M_i^2)$. The total area of $O(N_b / \log N_b)$ residue digits is $\approx O((N_b / \log N_b) \cdot (\log N_b)^2) = O(N_b \log N_b)$.

In the binary to residue conversion part, a Booth (or Booth-like) algorithm can be used to reduce the number of adder stages by pairing the high-order control bits. For instance, bit j, and j+1 as well as the carry bit generated from the last conversion stage can be used to decide which of the 8 constants to be added.

j	j+1	carry	constant to be added
0	0	0	$0 \bmod m_i$
0	0	1	$2^{M_i} \bmod m_i$
0	1	0	$2^{j+1} \bmod m_i$
0	1	1	$2^{j+1} + 2^{M_i} \bmod m_i$
1	0	0	$2^j \bmod m_i$
1	0	1	$2^j + 2^{M_i} \bmod m_i$
1	1	0	$2^j + 2^{j+1} \bmod m_i$
1	1	1	$2^j + 2^{j+1} + 2^{M_i} \bmod m_i$

However, in each stage a few constants need to be stored. It is a trade-off as to how many bits should be grouped in one stage. As a rule of thumb, the number of constants in each level of the adder should not exceed 8, therefore no more than 2 bits are to be grouped in this scheme.



x_i
y_i
BINARY MULTIPLIER
$2M_i - 1$
j
M_i
$M_i - 1$
0
bit number
MUX &ADD
a
b
MUX &ADD
ab=00 $\longrightarrow$ +0
ab=01 $\longrightarrow$ $+2^j \bmod m_i$
ab=10 $\longrightarrow$ $+2^{M_i} \bmod m_i$
ab=11 $\longrightarrow$ $+2^j + 2^{M_i} \bmod m_i$
carry
MUX &ADD
LAST STAGE CONVERSION
z_i

Figure 5. A residue array multiplier

3.3 Summary of Area Complexities of Different Multiplier Schemes

Some features of the four residue multiplier designs discussed above are summarized in Table 4. From this table it is clear that the quarter square approach is of limited interest, and that each of the other approaches may be superior to the rest depending on the value of M_i. To determine performance characteristics as well as more precise area requirements, and their dependence on feature sizes, some sample designs were made in nMOS.

	Table(bits)	1-bit Bin. Adder	1-bit Res. Adder
TABLE	$M_i 4^{M_i}$	0	0
QUARTER SQUARE	$4M_i 2^{M_i}$	M_i	M_i
INDEX TRANSFORM	$3M_i 2^{M_i}$	0	M_i
ARRAY	0	$2M_i^2$	0

	Table(bits)	1-bit Bin. Adder	1-bit Res. Adder
TABLE	$N_b{}^3$	0	0
QUARTER SQUARE	$4N_b{}^2$	N_r	N_r
INDEX TRANSFORM	$3N_b{}^2$	0	N_r
ARRAY	0	$2N_b \log N_b$	0

Table 4. Some features of residue multipliers for one residue digit (top) and the whole system (bottom)

4. Residue Multipliers in nMOS

4.1 Delay and Area Estimates

In the last section, different schemes for implementing residue multipliers and their relative complexities were discussed in terms of order statistics. For most applications, more accurate performance evaluations are necessary. Since different residue multipliers are composed of tables, adders, or a combination of both tables and adders, the total delay time and chip area can be calculated by adding the delay and area of the basic building components - tables and adders.

In general, tables are implemented by PLAs. In [Tr83], an equation is derived to calculate the chip area of MOS PLAs in terms of the number of inputs, minterms and outputs. To estimate the delay of a MOS PLA, a model based on an extension of the τ-model [Me80] is developed in the second part of this report. The worst case delay is expressed in terms of the number of inputs, minterms, and outputs (see eq. 12 in Part II).

The delay time and the chip area of various adders may differ significantly due to the great diversity of designs. For a reasonable trade-off between delay time and chip area, most adders are in the area range $3000\lambda^2 \sim 5000\lambda^2$ and with a delay of $40\tau \sim 60\tau$. To first order a one-bit adder has area $= 4000\lambda^2$ and delay$= 50\tau$.

We have derived a method to calculate the delay and area of tables and adders. Therefore the delay time and chip area of different residue multipliers is easily obtained since the size of the basic building block in a residue multiplier is known. Figure 6 and 7 contain delay estimates for residue multiplier designs in

4μm and 0.5μm nMOS.

The delays are computed in a manner illustrated by the following examples taken from Figure 6. We calculate the delays in three different 5×5 residue multipliers as specified in the third row.

1. Table look-up multiplier:

The table is implemented by a PLA with 10 inputs and 5 outputs. There are as many as $2^{10} = 1024$ product terms (rows) in the AND-plane. On the other hand, the OR-plane has only 5 columns. The large number of product terms implies a long wire delay. In order to minimize this delay, the bit pattern can be reorganized so that the PLA has relatively balanced AND- and OR-plane (see part 2, section 4, subsection "How to improve the performance" item 2). In this particular 10-input 5-output PLA, if 6 bits are used for the inputs to the AND-plane, then the number of product terms is reduced to 64. However, the OR-plane will generate 80 outputs. Only 5 out of these 80 outputs are selected in a multiplexor which uses the remaining 4 input bits as control bits ($2^4 = 16 = 80/5$). In this configuration, the actual number of inputs, product terms and outputs used in the delay equation (eq.12 part II) are I=6, P=64, O=80 respectively. Using the values of I, P, and O in the delay equation gives the delay $\approx 923\tau$

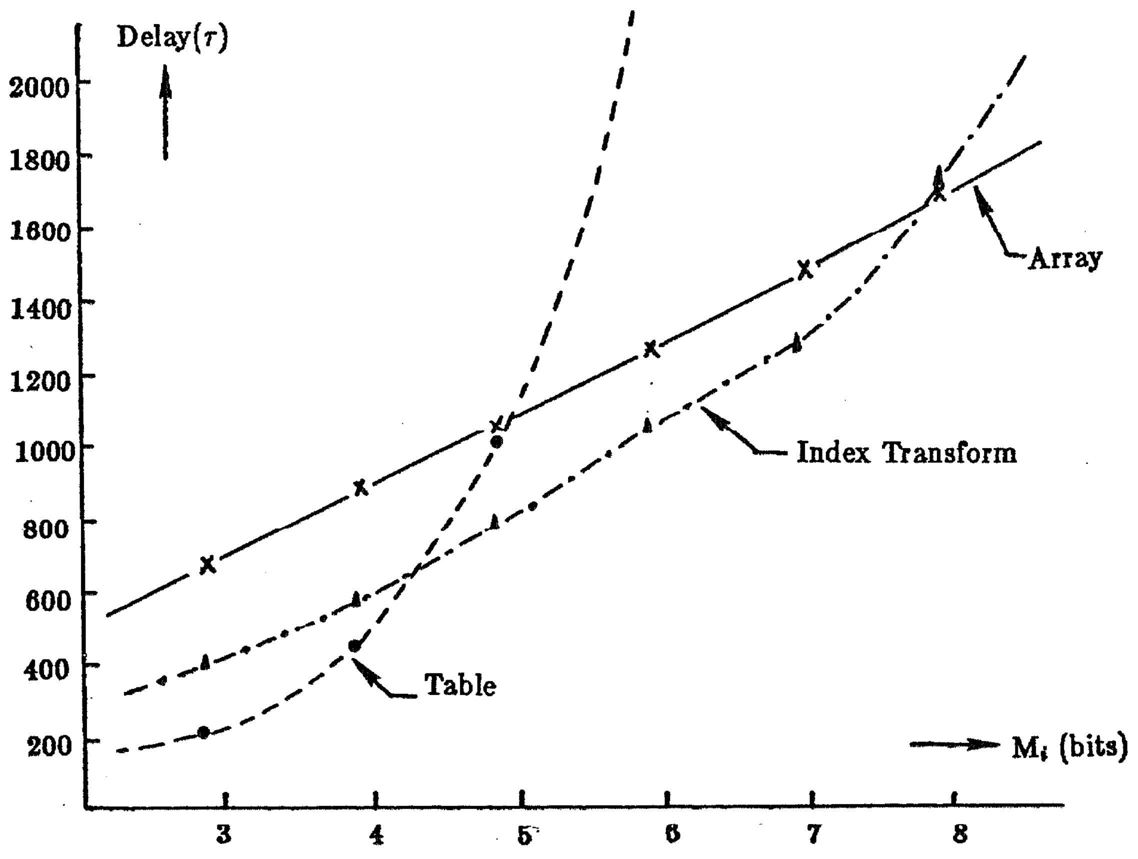
2. Index Transform Multiplier

The delay in a 5×5 index transform residue multiplier is composed of the delays in the INDEX transform table, the mod$(m_i - 1)$ residue adder and the INVERSE transform table. The INDEX and INVERSE INDEX transform tables are implemented by PLA's with 5 inputs and 5 outputs. To make the table square like, the actual number of inputs is 4, the number of product terms is 16 and the number of outputs is 10. Using the values I=4, P=16 and O=10 in the delay equation gives a table delay of $\approx 159\tau$ The critical path in the 5-bit residue adder consists of 6 1-bit adders. This is a delay of $\approx 6 \times 50\tau = 300\tau$. The total delay is $\approx 159 \times 2 + 300 \approx 618\tau$.

3. Array Multiplier

The signal in a critical path propagates through $4 \times 5 = 20$ adders. Therefore the delay is $\approx 20 \times 50\tau = 1000\tau$.

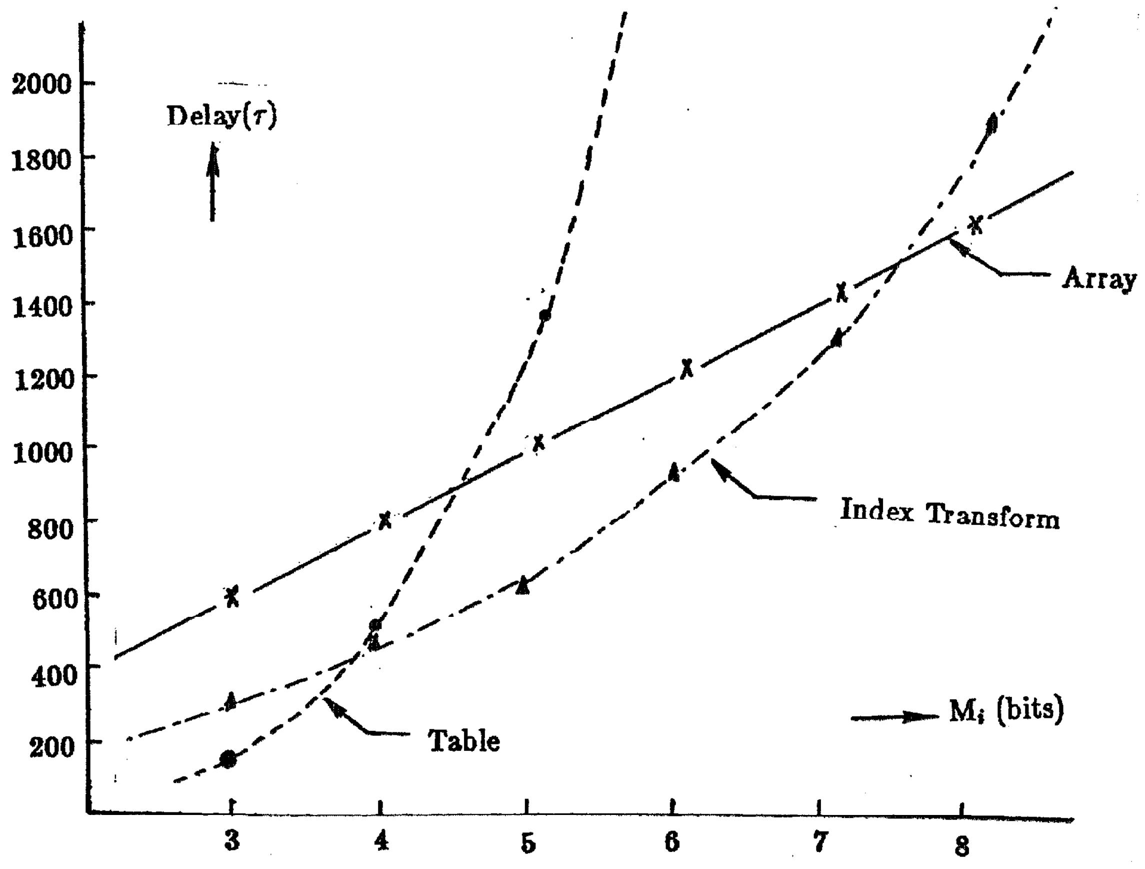
As is apparent from the figures, table look-up does not scale well if the storage is unstructured. The delay decreases only slightly. When measured in τ it increases significantly. The index transform approach employs much smaller tables. The delay measured in τ increases only slightly for $M \leq 8$ bits, i.e., the delay is reduced almost to the same extent as the switching speed of the devices while the feature sizes are reduced. The areas, see Figure 8, expressed in terms of λ^2 [Me80] is to first order independent of the feature size.



Number of bits	Table	Ind. tran.	Array
3	173	331	600
4	396	454	800
5	923	618	1000
6	2220	858	1200
7	5724	1137	1400
8	16553	1602	1600

(unit : τ $1\tau \approx 0.2$ns)

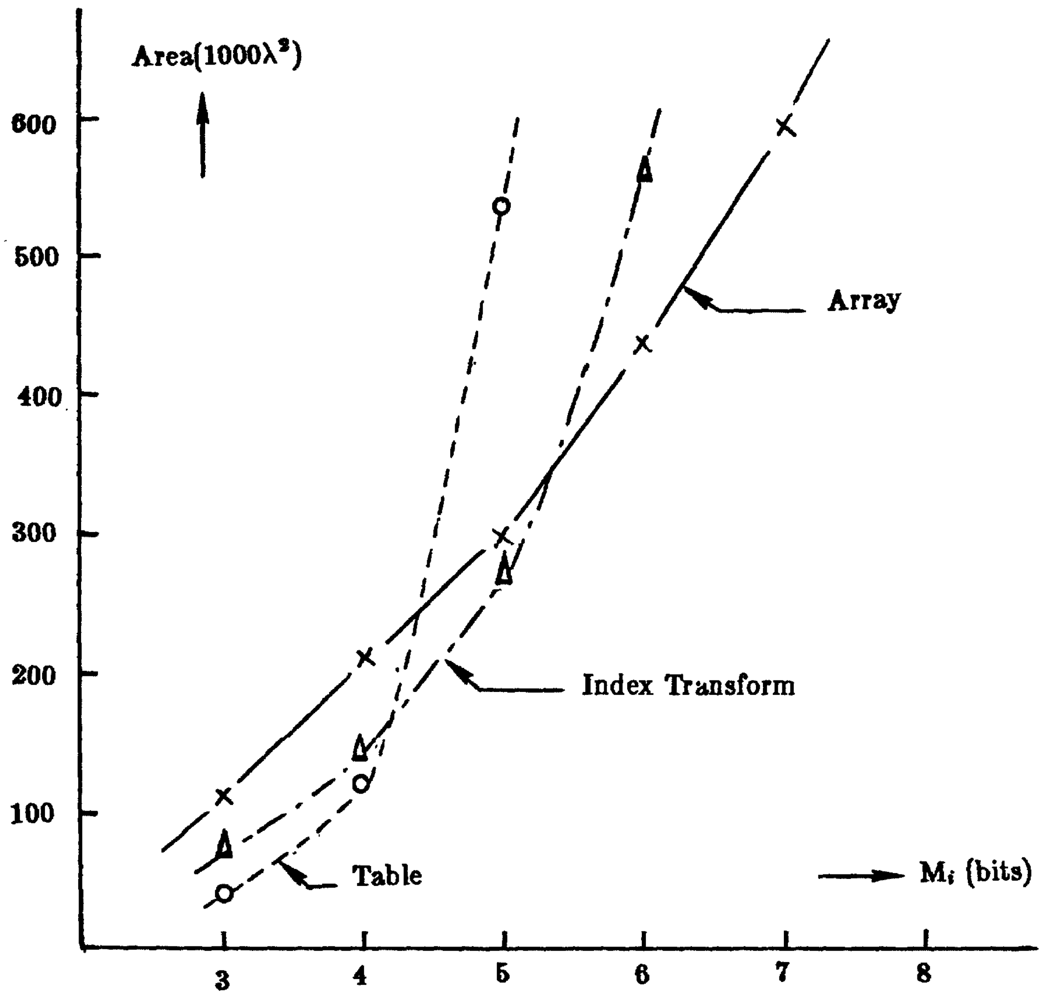
Figure 6. Delay in residue multipliers for 4μm nMOS



Number of bits	Table	Ind. tran.	Array
3	188	335	600
4	475	464	800
5	1327	646	1000
6	4270	922	1200
7	15974	1277	1400
8	67015	1997	1600

(unit : τ $1\tau \approx 0.025$ns)

Figure 7. Delay in residue multipliers for 0.5μm nMOS



Number of bits	Table	Ind. tran.	Array
3	45	95	115
4	147	168	207
5	524	282	297
6	1970	543	407
7	8000	1007	591
8	32000	2177	664

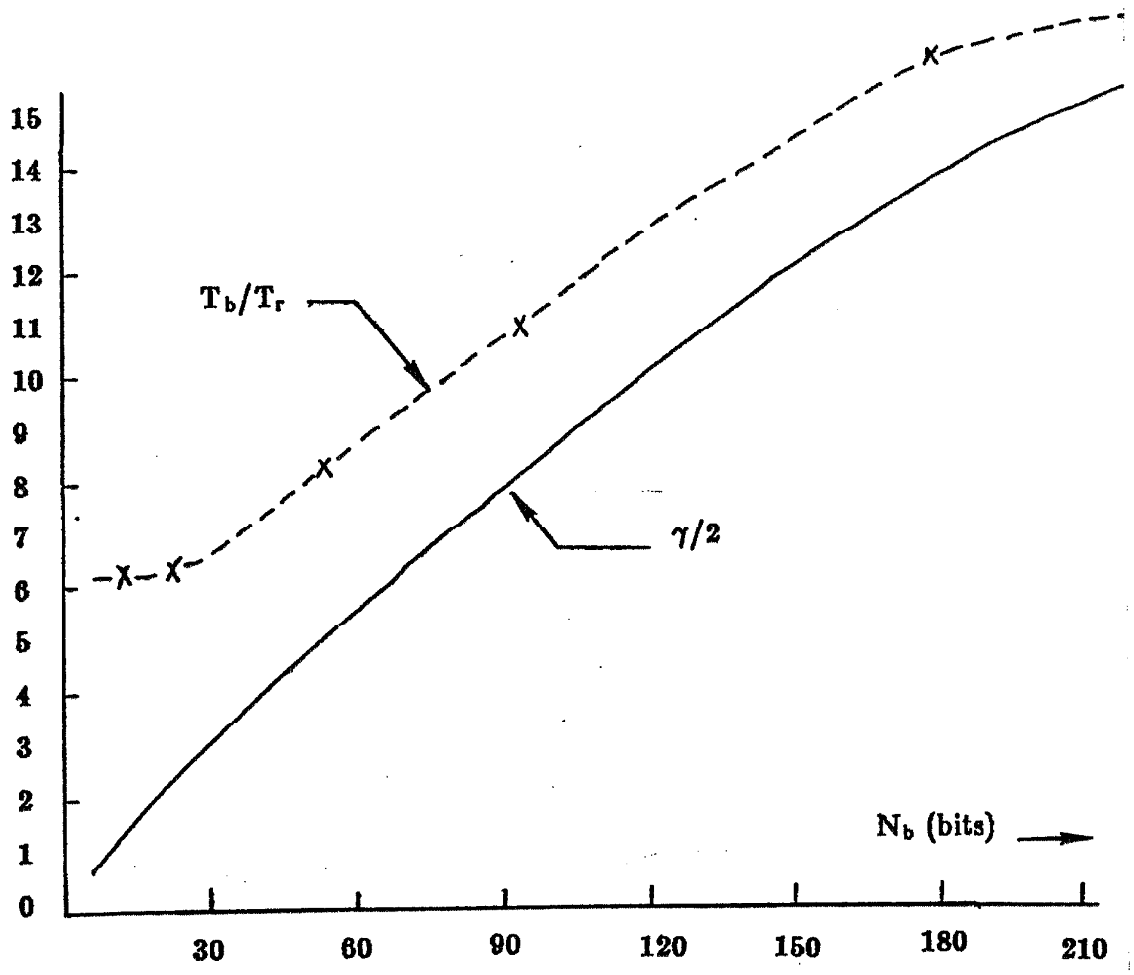
(unit : 1000λ^2)

Figure 8. Chip area of residue multipliers

4.2 Performance Comparison between Residue Arithmetic and Binary Arithmetic

Figure 9 shows the relative speeds of multiplying two N_b-bit numbers in the residue number system (delay$=T_r$) and the binary number system (delay$=T_b$). T_r is taken to be the minimum delay of the three alternative designs investigated, i.e., table look-up, index transform, and array multiplier with conversion. The values for T_r are obtained from Figure 6. T_b is obtained by assuming that a binary array multiplier is used. An $N_b \times N_b$ array multiplier has a delay time $T_b = 2N_b \times$(1-bit adder delay).

For $M > 7$ the array scheme is the most effective way to implement a residue multiplier. The speed-up ratio T_b/T_r between residue arithmetic and binary arithmetic is of $O(N_b/\log N_b)$. In a residue array multiplier, the binary to residue conversion part and the binary multiplication part have the same delay. Therefore, $T_b/T_r \approx \gamma/2$. For a relatively small range R, the actual speed-up ratio is better than $\gamma/2$. The fact that the speed-up is better than the asymptotic ratio for a small range is due to the use of different implementation techniques. Table look-up is efficient for a small range. For a large range residue multiplication as well as binary multiplication make use of array multipliers. Roughly speaking, the speed for multiplication in the residue number system is about 5 to 10 times higher than the speed in the binary number system for a moderate range.



N_b	N_r	M	γ	T_r	T_b	T_b/T_r
10	12	3	3.3	173	1000	5.8
20	22	4	5.0	396	2000	5.1
48	51	5	9.0	618	4800	7.8
90	97	6	15.0	858	9000	10.4
184	196	7	26.3	1137	18400	16.2

(4μm nMOS technology)

Figure 9. Comparison between residue and binary arithmetic

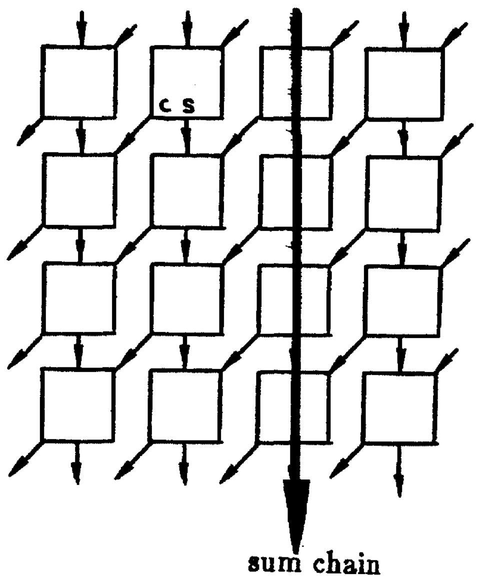
4.3 Chip Characteristics

A parameterized residue array multiplier has been designed. The multiplier performs $M_i \times M_i$ bit residue multiplication with respect to modulus m_i. The design is composed of two parts: a $M_i \times M_i$ binary multiplier that produces a $2M_i$ bit result, and a binary to residue converter which converts a $2M_i$ bit binary number to a M_i bit residue number (with respect to the modulus m_i). Both parts are composed of arrays of adders.

A Binary Multiplier with Carry Save Adders

The binary multiplier is composed of arrays of carry save adders. In a multiplier which consists of carry save adders, there are two critical paths: the carry chain and the sum chain, Figure 10. The speed of generating the sum bit should be as fast as the speed of generating the carry bit. A Manchester carry chain as presented in [Me80] enhances the speed of carry bit generation at the expense of the speed of sum bit generation.

Figure 11 shows the logic diagram of a nMOS carry save adder. The layout is in Figure 12. Circuit simulation results indicate that the delay for generating sum bits and the carry bit are about the same ($8 \approx 10$ ns for 4 μm technology), see Figure 13.



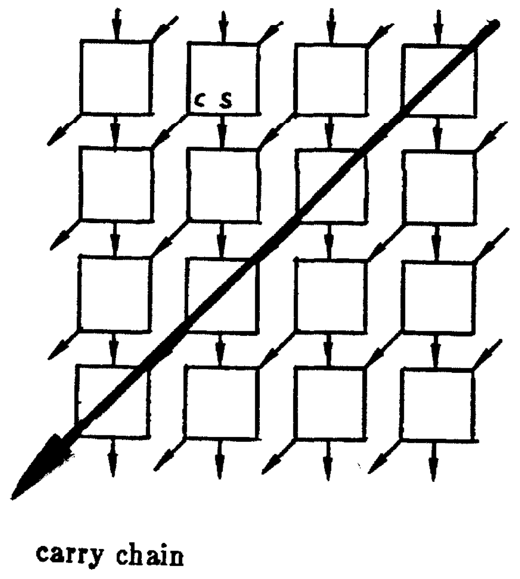


Figure 10. Sum chain and carry chain

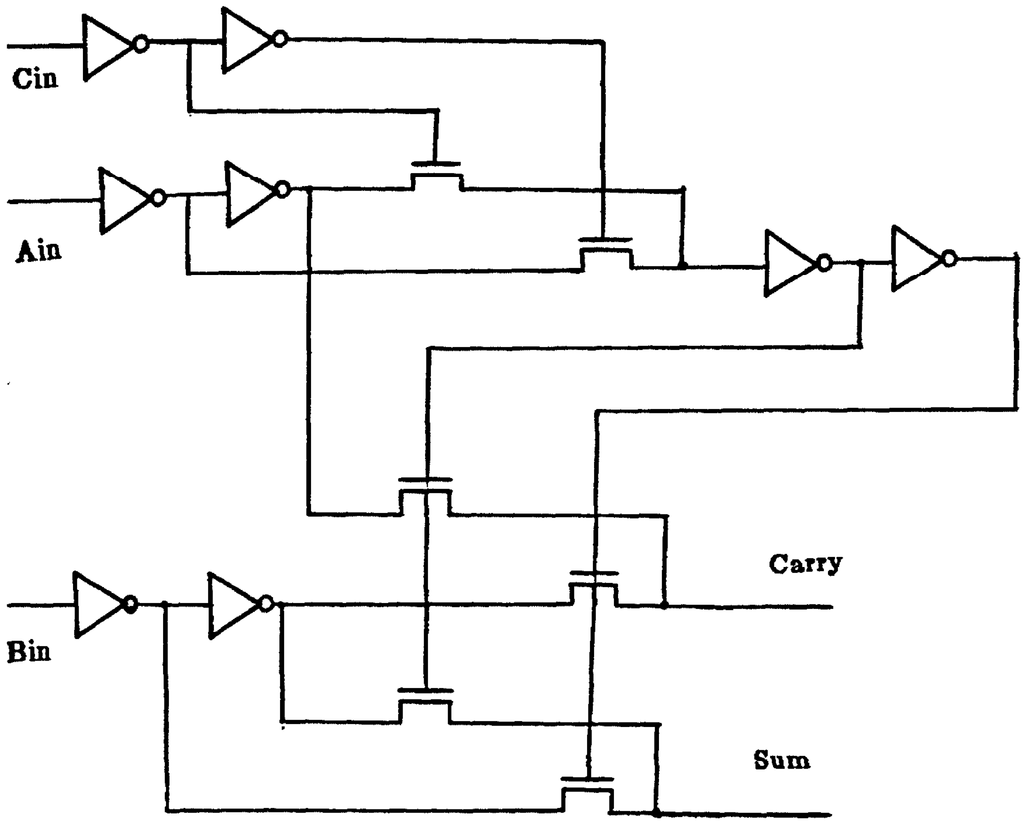


Figure 11. The logic diagram of the carry save adder

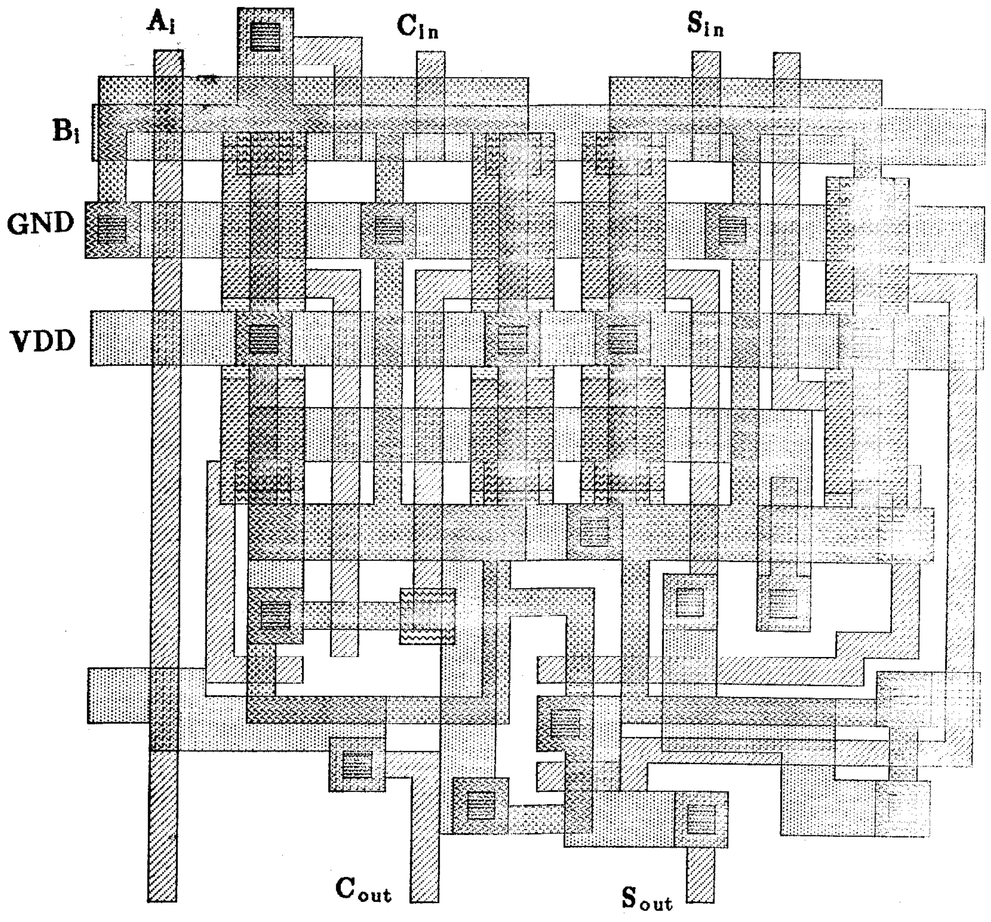
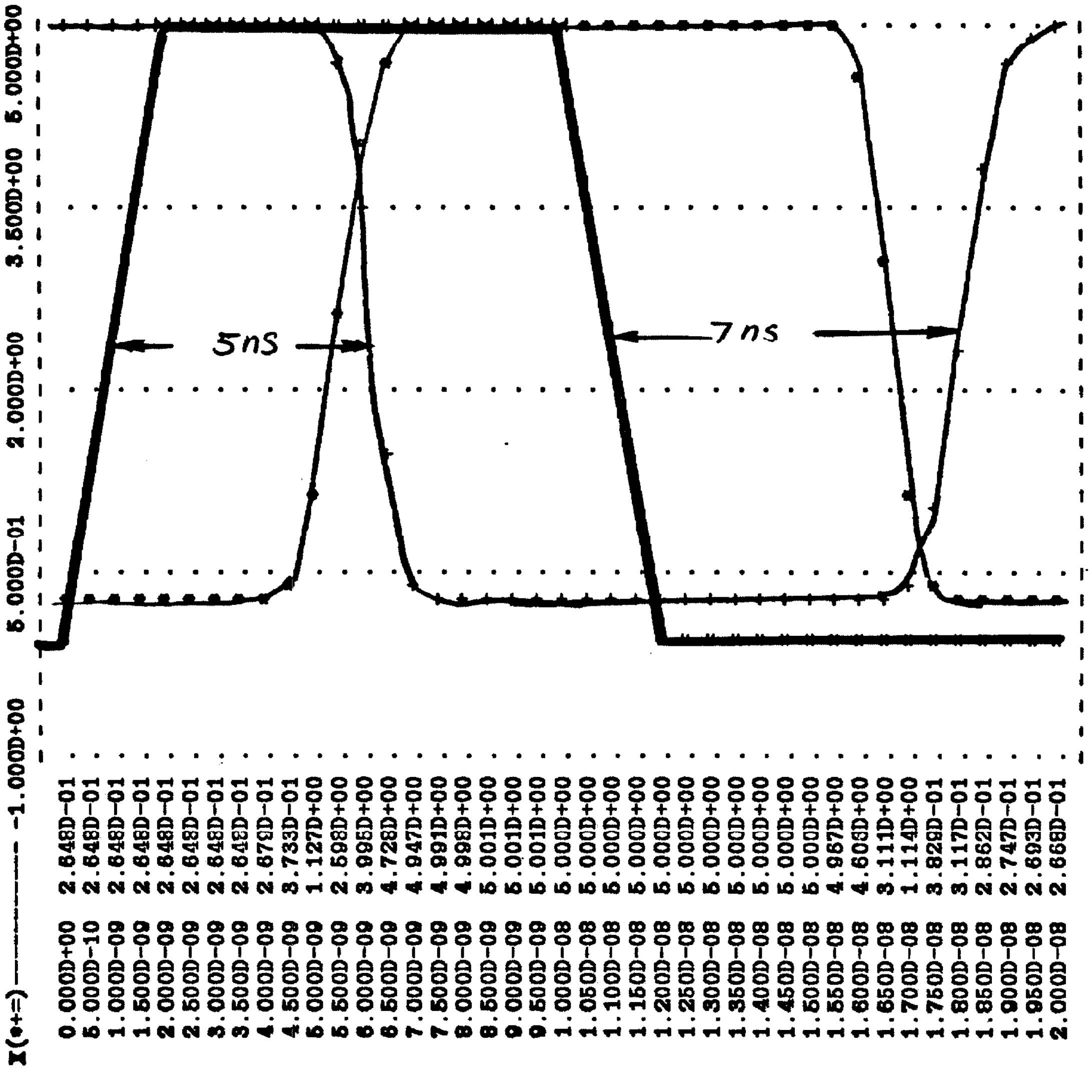


Figure 12. The layout of the carry save adder



```
X(*+=)---------- -1.000D+00
0.000D+00   2.648D-01
5.000D-10   2.648D-01
1.000D-09   2.648D-01
1.500D-09   2.648D-01
2.000D-09   2.648D-01
2.500D-09   2.648D-01
3.000D-09   2.648D-01
3.500D-09   2.648D-01
4.000D-09   2.679D-01
4.500D-09   3.733D-01
5.000D-09   1.127D+00
5.500D-09   2.598D+00
6.000D-09   3.995D+00
6.500D-09   4.728D+00
7.000D-09   4.947D+00
7.500D-09   4.991D+00
8.000D-09   4.998D+00
8.500D-09   5.001D+00
9.000D-09   5.001D+00
9.500D-09   5.001D+00
1.000D-08   5.000D+00
1.050D-08   5.000D+00
1.100D-08   5.000D+00
1.150D-08   5.000D+00
1.200D-08   5.000D+00
1.250D-08   5.000D+00
1.300D-08   5.000D+00
1.350D-08   5.000D+00
1.400D-08   5.000D+00
1.450D-08   5.000D+00
1.500D-08   5.000D+00
1.550D-08   4.967D+00
1.600D-08   4.608D+00
1.650D-08   3.111D+00
1.700D-08   1.114D+00
1.750D-08   3.829D-01
1.800D-08   3.117D-01
1.850D-08   2.862D-01
1.900D-08   2.747D-01
1.950D-08   2.693D-01
2.000D-08   2.668D-01
```

Figure 13. The circuit simulation result of the carry save adder

Binary to Residue Conversion Using Multiplexed Adders

The binary to residue conversion part is composed of M_i stages of adders. In each stage, one of the four constants $\underline{0}$, $\underline{2^j \bmod m_i}$, $\underline{2^{M_i} \bmod m_i}$, $\underline{2^j + 2^{M_i} \bmod m_i}$ is added to the partial sum, where j indicates the stage number from M_i to $2M_i - 1$ (see figure 5). Which constant should be added is determined by two control bits. One is the j-th bit of the number generated by the binary array multiplier. The other is the carry bit generated by the previous stage or adder. Notice that the constants and the partial sums (not including the carry bit) are all M_i-bit numbers.

The basic cell in the conversion part is a multiplexed adder. It has two inputs and four control signals: The two inputs, S_{in} and C_{in}, come from the adder in the previous stage. The two control signals that determines which one of the four constants should be added, are decoded into the control signals m_1, m_2, m_3 and m_4.

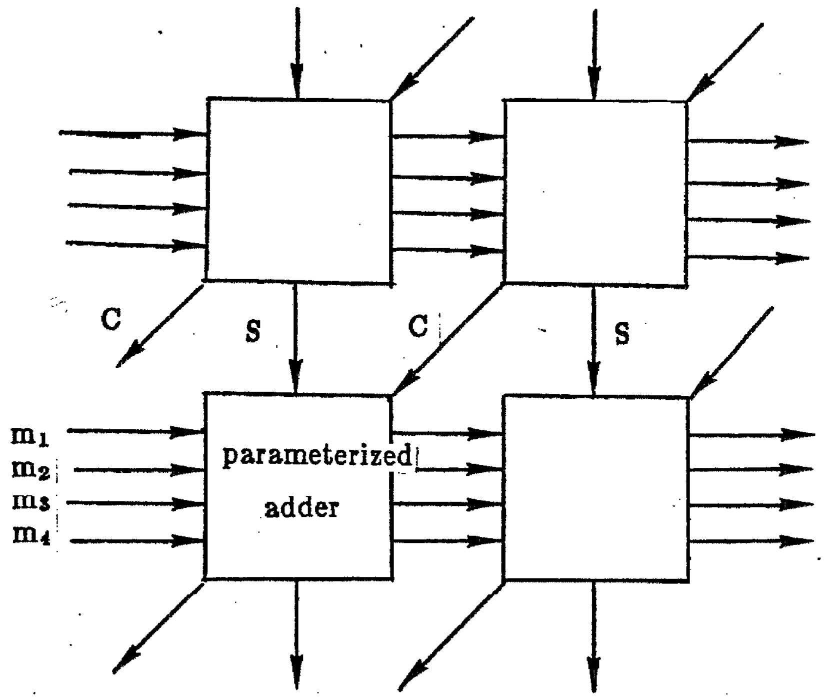


Figure 14. The multiplexed adder

The multiplexed adder can also be considered as a 3-input full adder. Two of the inputs are S_{in} and C_{in}. The third input is a function of the control signals. If m_1 is 1 then the third input is always 0. If m_2 is 1 then the third input may be either 0 or 1, depending on the constant $2^j \bmod m_i$, as is the case when m_3 is 1 or m_4 is 1. Therefore there are 8 different combinations:

control signal	the third input bit
m_1=1	0 0 0 0 0 0 0 0
m_2=1	0 0 0 0 1 1 1 1
m_3=1	0 0 1 1 0 0 1 1
m_4=1	0 1 0 1 0 1 0 1

The third input bit pattern can be used to parameterize the multiplexed adder cells. The parameterized adder cell is composed of two parts: a two-input half adder and a multiplexor. The half adder takes inputs S_{in} and C_{in}, and generates four outputs c_0, s_0, c_1 and s_1. c_0 and s_0 are the carry bit and sum bit generated as if the third input is 0. c_1 and s_1 are the carry bit and sum bit generated as if the third bit is 1.

C_{in}	S_{in}	c_0	s_0	c_1	s_1
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	0	1	1

It can be easily verified that c_0=AND(C_{in},S_{in}), c_1=OR(C_{in},S_{in}), s_0=XOR(C_{in},S_{in}) and s_1=XNOR(C_{in},S_{in}).

The multiplexor takes c_0, s_0, c_1, s_1 as inputs and uses four control signals m_1,m_2,m_3 and m_4 to determine whether the output is (c_0,s_0) or (c_1,s_1).

For example, if the third input bit pattern is (0101), then the multiplexor will perform the following function:

$$\text{if } (m_1,m_2,m_3,m_4)=(1,0,0,0) \text{ then } C_{out}=\mathbf{MAJ}(C_{in},S_{in},0)=c_0$$
$$S_{out}=\mathbf{XOR}(C_{in},S_{in},0)=s_0$$
$$\text{if } (m_1,m_2,m_3,m_4)=(0,1,0,0) \text{ then } C_{out}=\mathbf{MAJ}(C_{in},S_{in},1)=c_1$$
$$S_{out}=\mathbf{XOR}(C_{in},S_{in},1)=s_1$$
$$\text{if } (m_1,m_2,m_3,m_4)=(0,0,1,0) \text{ then } C_{out}=\mathbf{MAJ}(C_{in},S_{in},0)=c_0$$
$$S_{out}=\mathbf{XOR}(C_{in},S_{in},0)=s_0$$
$$\text{if } (m_1,m_2,m_3,m_4)=(0,0,0,1) \text{ then } C_{out}=\mathbf{MAJ}(C_{in},S_{in},1)=c_1$$
$$S_{out}=\mathbf{XOR}(C_{in},S_{in},1)=s_1$$

(**MAJ** is the majority function and **XOR** is the exclusive-or function)

Express C_{out} and S_{out} as boolean functions:

$$S_{out}=m_1m_2'm_3'm_4's_0+m_1'm_2m_3'm_4's_1+m_1'm_2'm_3m_4's_0+m_1'm_2'm_3'm_4s_1$$
$$C_{out}=m_1m_2'm_3'm_4'c_0+m_1'm_2m_3'm_4'c_1+m_1'm_2'm_3m_4'c_0+m_1'm_2'm_3'm_4c_1$$

Figure 15 shows the logic diagram of the (0101) multiplexed adder. The layout is in Figure 16. The multiplexor parts of the 8 different parameterized cells are shown in Figure 17.

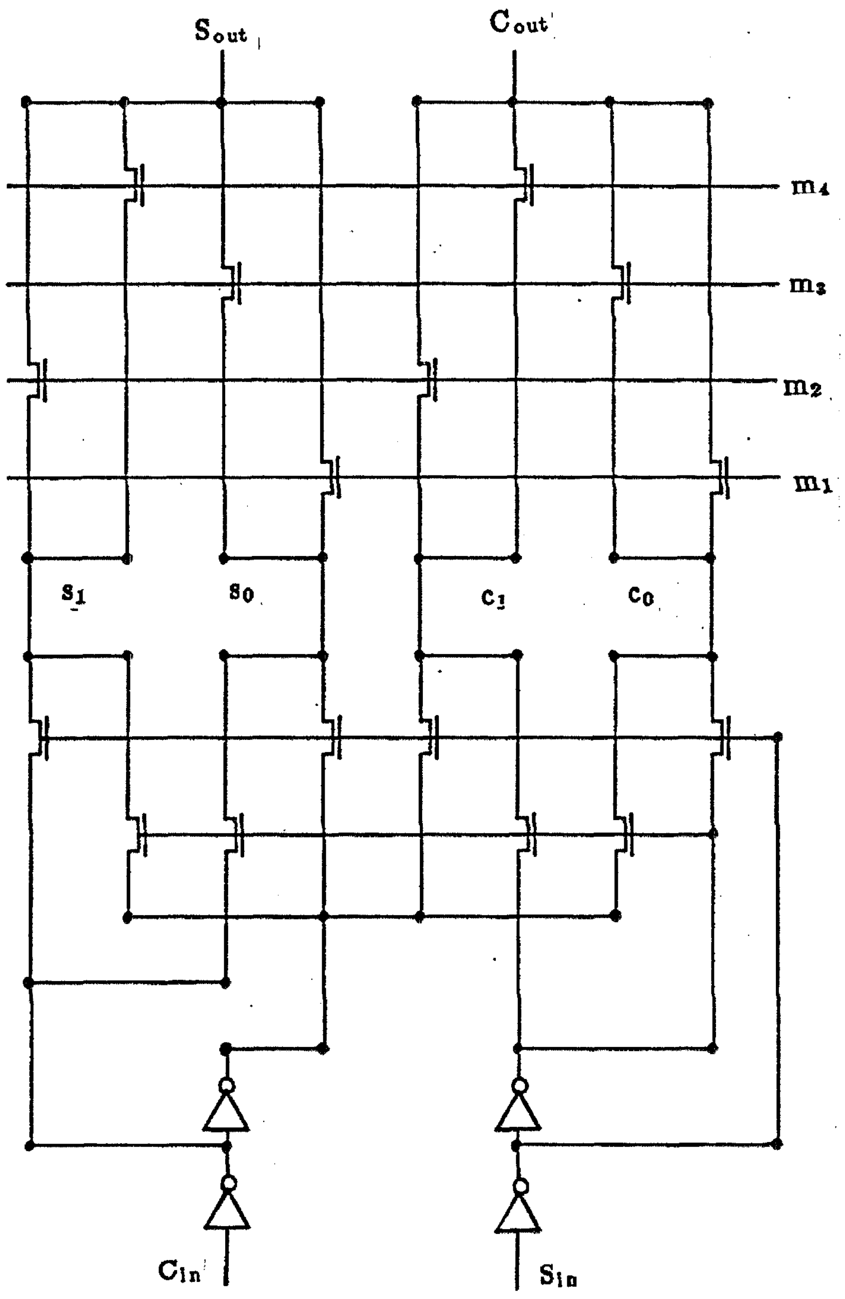


Figure 15. The logic diagram of a (0101) multiplexed adder

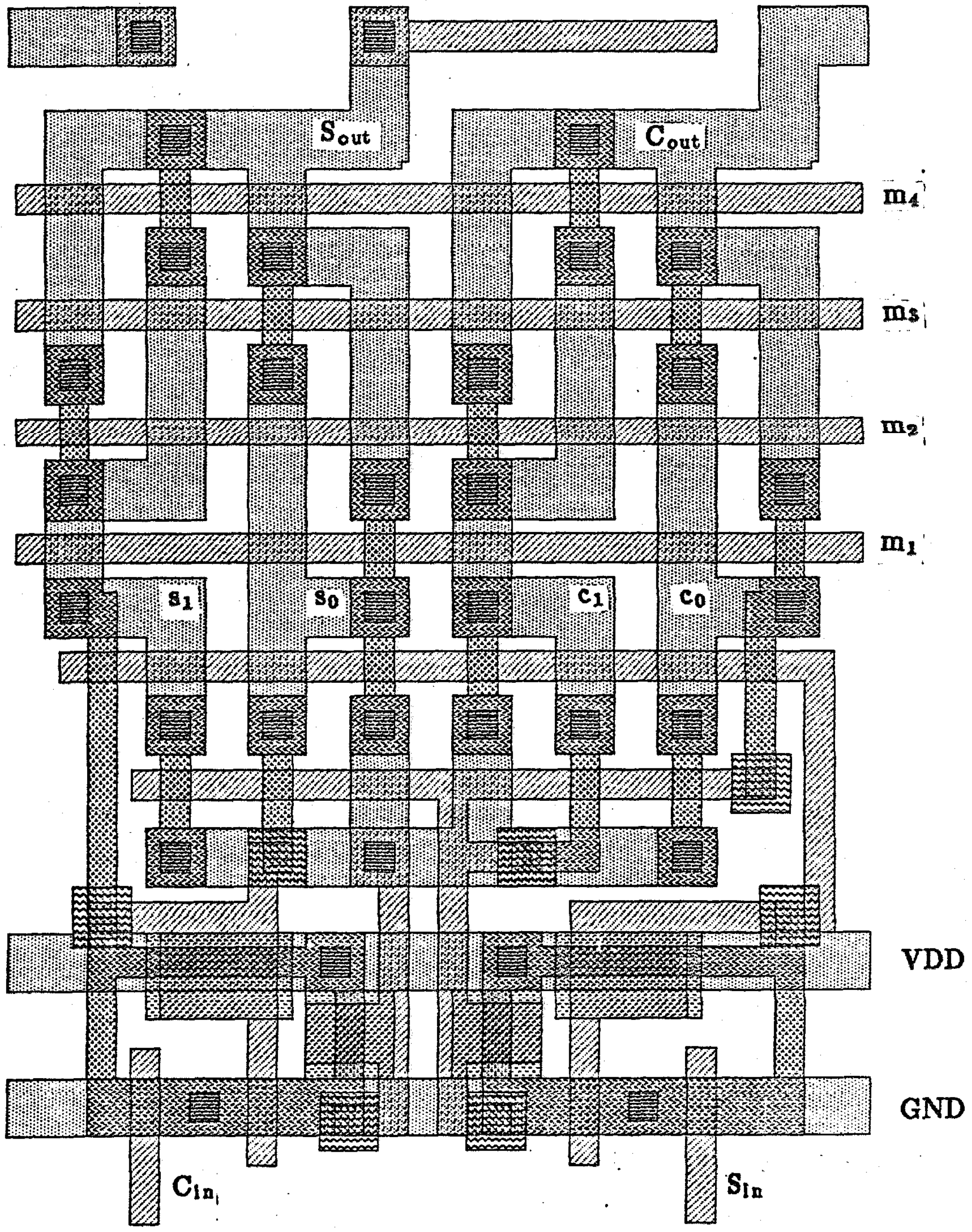


Figure 16. The layout of a (010i) multiplexed adder

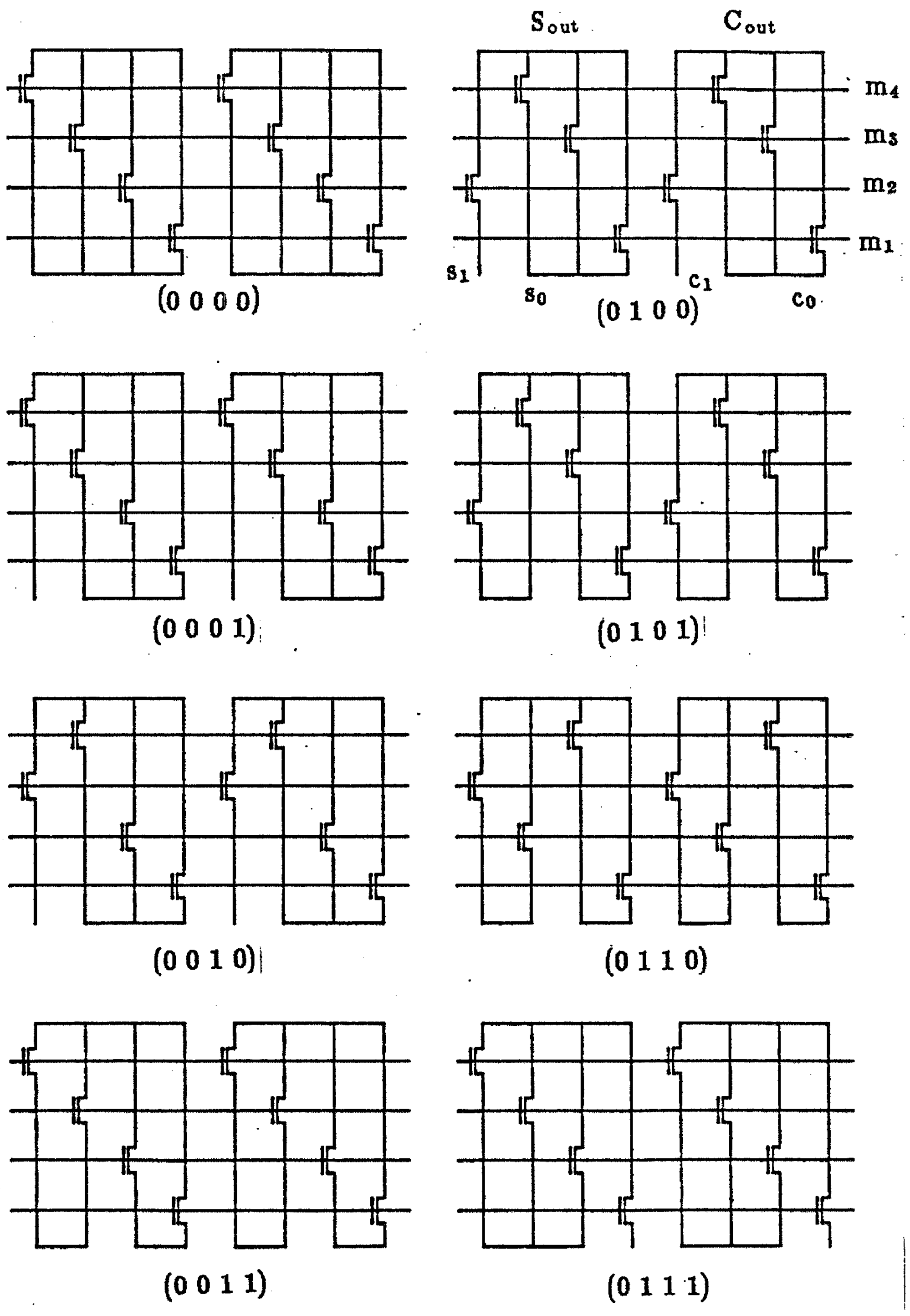


Figure 17. The multiplexor part of 8 different parameterized adder cells

The layout of the whole chip is generated by a program written in Earl code [Ki82]. The cells are parameterized so that the program can generate the layout for different moduli. Figure 18 shows the metal layer of the layout of an 8-bit residue array multiplier. A 4-bit version of the multiplier requires an area of $300 \times 700\lambda^2$, and an 8-bit version $540 \times 1200\lambda^2$.

Simulations indicate that carry and sum bit generation requires 8ns$\sim$10ns. The critical path in each M_i-bit residue multiplier goes through $4M_i$ adders. Therefore, the total delay is $32M_i$ to $40M_i$ ns, and hence about 250$\sim$320 ns for an 8-bit multiplier.

The performance of the design can be improved in several ways. For instance, the sample design does not have any form of pipelining. Optimizing the residue array multiplier design should make it competitive for smaller values of M_i.

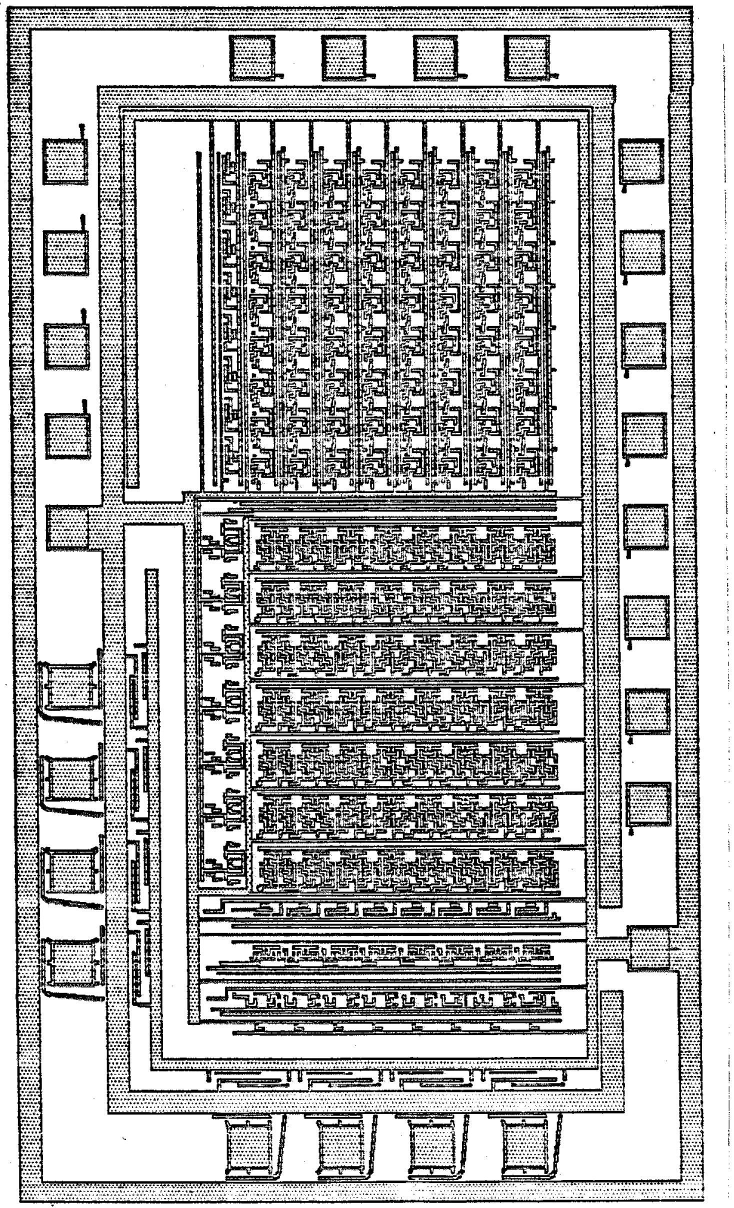


Figure 18. The metal layer of an 8-bit residue multiplier

5. Base Selection and Design Cost

The cost of a VLSI design is affected by many factors such as chip area, speed, layout regularity and ease of design. There are some trade-offs between these factors to minimize the design cost. In residue arithmetic, total design cost depends on the individual cost of each digit. Therefore base selection is important for the total cost. This section contains two heuristic algorithms for base selection with respect to minimum delay, or minimum area.

Problem 1:

Given a cost function $T(n)$ (delay) and range requirement R ($\lceil \log R \rceil = N_b$ bits), find a base $S = \{m_1, \ldots, m_p\}$ such that $\log(\prod_{i=1}^{p} m_i) \geq N_b$ and $\max\{T(m_i)\}$ is minimized.

Algorithm 1:

```
1.  BEGIN
2.  find $n_1$ such that $T(n_1)$ is minimum, set $S \leftarrow \{n_1\}$
3.  WHILE   (range of S)$< N_b$   DO
4.      BEGIN   { add/substitute the next modulus }
5.      For all $k$ not examined,
6.          $S_k \leftarrow S \bigcup \{k\} - \{n_j \mid n_j \in S, \gcd(k, n_j) \neq 1\}$
7.          $\Delta R_k$ =(range of $S_k$)− (range of S),
8.      Select $S_k$ such that $\Delta R > 0$ and $T(k)$ minimum.
        If there are more than one such $S_k$,
        select the one whose $\Delta R_k$ is maximum.
9.      Set $S \leftarrow S_k$
10.     END
11. END
12. S is the base wanted
```

In algorithm 1, the base S starts with a modulus with overall minimum cost (line 2) and then searches for the next larger modulus until the range requirement is met (line 3 to line 10). The next modulus selected must (1) be relatively prime with respect to other moduli already in S (line 5), and (2) have the smallest cost among all possible candidates (line 8).

Problem 2:
Given the cost function $A(n)$ (area) and the range requirement R ($N_b = \lceil \log R \rceil$ bit), find a base $S = \{m_1, \ldots, m_p\}$ such that $\log(\prod_{i=1}^{p} m_i) \geq N_b$ and $\sum_i A(m_i)$ is minimized.

Algorithm 2:

```
1.  BEGIN
2.  find $n_1$ such that $A(n)$ is minimum at $n_1$, set $S \leftarrow \{n_1\}$
3.  WHILE   (range of S)$<$ R    DO
4.          BEGIN { add/substitute the next modulus }
5.          for all $k$ not examined
6.                  $S_k \leftarrow S \bigcup \{k\} - \{n_j \mid n_j \in S, \gcd(k, n_j) \neq 1\}$
7.                  $\Delta R_k$ =(range of $S_k$)$-$ (range of S),
8.                  $\Delta A_k$ =(total area of $S_k$) $-$ (total area of S)
9.          select $S_k$ such that $\frac{\Delta A_k}{\Delta R_k}$ is minimum and set $S \leftarrow S_k$
10.         END
11. END
12. S is the base wanted
```

In algorithm 2, as in algorithm 1, the base selection starts with finding a modulus with overall minimum cost (line 2) and then searches for the next modulus until the range requirement is met (line 3 to line 10). The requirements for the next modulus selected are somewhat different from those in algorithm 1. The relatively prime requirement is still the same. The second requirement is sharply different, it becomes: a selected number should have minimum $\frac{\Delta A_k}{\Delta R_k}$ among all candidates. The reason for this criterion is due to the fact that the chip area is additive. To make a minimal total area, the area incremental rate should be kept as small as possible.

The purpose of the following examples are to illustrate the construction of an optimal base with respect to minimum delay and minimum area respectively. The cost functions for delay and area used in these examples are obtained from Figure 6 and Figure 8. It should be noted that a modulo of the form 2^k has lower time and area cost than other moduli for every implementation scheme studied. In an array implementation no conversion is necessary. In a table implementation the PLA will have fewer product terms after a Boolean function minimization is carried out.

A modulo $2^k - 1$ or $2^k + 1$ multiplier has lower time and area cost when implemented by arrays. The modulo computation does not require a large conversion section. However, in a table look-up scheme no reduction in table size is possible. It should be noticed though, that a modulo of the form $2^k - 1$ makes more efficient use of the storage, since a k-bit wide table suffice, whereas modulo $2^k + 1$ requires k+1 bits.

In the example of the base selection algorithm that follows it should be noticed that the range is small enough that table look-up is the optimum way of implementing the residue multiplier. The moduli require less than 6-bits for their binary encoding. Hence, moduli in the interval between moduli of the form 2^k require about equal time and area. However, a modulo $257(2^8 + 1)$ residue multiplier require less area than a modulo 259 residue multiplier, since such a multiplier would be implemented as an array multiplier and the latter modulus would require a conversion stage.

EXAMPLE 1. Assume that the required range $R=3 \times 10^{10}$ ($N_b = 34.8$ bits). The goal is to find a base with range $\geq R$ and the smallest delay time. Assume the delay time function $T(n)$ of the first few numbers have the values listed below.

n	T(n)
2	20
3	63
4	50
5~7	173
8	90
9~15	396
16	185
17~31	618
32	330
33~63	858
64	570

Algorithm 1 gives the following result:

Action	Base (S)	Range(bit)	Cost (Time)
(1) $S \leftarrow \{2\}$	$\{2\}$	1	20
(2) $S \leftarrow S \bigcup \{4\} - \{2\}$	$\{4\}$	2	50
(3) $S \leftarrow S \bigcup \{3\}$	$\{4, 3\}$	3.58	63
(4) $S \leftarrow S \bigcup \{8\} - \{4\}$	$\{8, 3\}$	4.58	90
(5) $S \leftarrow S \bigcup \{7\}$	$\{8, 3, 7\}$	7.39	173
(6) $S \leftarrow S \bigcup \{5\}$	$\{8, 3, 7, 5\}$	9.71	173
(7) $S \leftarrow S \bigcup \{16\} - \{8\}$	$\{16, 3, 7, 5\}$	10.71	185
(8) $S \leftarrow S \bigcup \{32\} - \{16\}$	$\{32, 3, 7, 5\}$	11.71	330
(9) $S \leftarrow S \bigcup \{13\}$	$\{32, 3, 7, 5, 13\}$	15.41	396
(10) $S \leftarrow S \bigcup \{11\}$	$\{32, 3, 7, 5, 13, 11\}$	18.87	396
(12) $S \leftarrow S \bigcup \{9\} - \{3\}$	$\{32, 9, 7, 5, 13, 11\}$	20.46	396
(13) $S \leftarrow S \bigcup \{64\} - \{32\}$	$\{64, 9, 7, 5, 13, 11\}$	21.46	570
(14) $S \leftarrow S \bigcup \{31\}$	$\{64, 9, 7, 5, 13, 11, 31\}$	26.41	618
(15) $S \leftarrow S \bigcup \{29\}$	$\{64, 9, 7, 5, 13, 11, 31, 29\}$	31.27	618
(16) $S \leftarrow S \bigcup \{27\} - \{9\}$	$\{64, 27, 7, 5, 13, 11, 31, 29\}$	32.85	618
(17) $S \leftarrow S \bigcup \{25\} - \{5\}$	$\{64, 27, 7, 25, 13, 11, 31, 29\}$	35.18	618

Let us consider action (7). The base S after action (6) is $\{8, 3, 7, 5\}$. The numbers already considered are 2,3,4,5,7,8. So,the next candidates are 6,9,10,11,12,13,-14,15,16,17...... Composite numbers such as 6,10,12,14,15 are of little interest because once these numbers are selected the resulting range will decrease, i.e., $\Delta R < 0$. The possible candidates left are 9,11,13,16,17, ... According to the delay function given, T(16)< T(9)= T(11)= T(13)< ... . Hence, T(16) is the smallest among all candidates. Therefore the next action is $S \leftarrow S \bigcup \{16\} - \{8\}$. In action (5), T(5)=T(7) but 7 has larger ΔR. Therefore 7 is selected. In action (9), 13 is selected instead of 11 for the same reason.

EXAMPLE 2. Assume that the required range R=6 $\times 10^{10}$ (N_b = 35.8 bits). The goal is to find a base with range $\geq$R and the smallest total area. Assume the area function A(n) of the first few numbers have the values listed below.

n	A(n)
2	10
3	25
4	20
5~7	45
8	35
9~15	147
16	80
17~31	282
32	160
33~63	407
64	320

Algorithm 2 gives the following result:

Action	Base (S)	$\Delta A/\Delta R$	Range(bit)	Area
(1) S$\leftarrow$ {2}	{2}	10.0	1.0	10
(2) S$\leftarrow$S$\bigcup${4} − {2}	{4}	10.0	2.0	20
(3) S$\leftarrow$S$\bigcup${8} − {4}	{8}	15.0	3.0	35
(4) S$\leftarrow$S$\bigcup${3}	{8, 3}	15.8	4.6	60
(5) S$\leftarrow$S$\bigcup${7}	{8, 7, 3}	16.0	6.7	105
(6) S$\leftarrow$S$\bigcup${5}	{8, 7, 3, 5}	19.4	9.7	150
(7) S$\leftarrow$S$\bigcup${13}	{8, 7, 3, 5, 13}	39.7	13.4	297
(8) S$\leftarrow$S$\bigcup${11}	{8, 7, 3, 5, 13, 11}	42.5	16.9	444
(9) S$\leftarrow$S$\bigcup${16}	{16, 7, 3, 5, 13, 11}	45.0	17.9	489
(10) S$\leftarrow$S$\bigcup${31}	{16, 7, 3, 5, 13, 11, 31}	56.9	22.8	771
(11) S$\leftarrow$S$\bigcup${29}	{16, 7, 3, 5, 13, 11, 31, 29}	58.0	27.7	1053
(12) S$\leftarrow$S$\bigcup${23}	{16, 7, 3, 5, 13, 11, 31, 29, 23}	62.3	32.2	1335
(13) S$\leftarrow$S$\bigcup${19}	{16, 7, 3, 5, 13, 11, 31, 29, 23, 19}	66.4	36.5	1617

As is shown in the above examples, both algorithms (1) and (2) are very efficient since only a small number of next candidates need to be tested. Indeed, it is only necessary to test numbers of the form p^k (p: prime) provided that the functions $\mathrm{A}(n)$ and $\mathrm{T}(n)$ satisfy the following relations:

$$\mathrm{T}(n_1 n_2) > \max\{\mathrm{T}(n_1), \mathrm{T}(n_2)\} \quad \text{and} \quad \mathrm{A}(n_1 n_2) > \mathrm{A}(n_1) + \mathrm{A}(n_2)$$

The relations above imply that the cost of an arithmetic unit for a residue digit with a composite modulus $n_1 n_2$ is greater than the total cost of an arithmetic unit with moduli n_1 and n_2, respectively. Since the cost functions $\mathrm{A}(n)$ and $\mathrm{T}(n)$ derived in section 4.1 satisfy the above relations, it is only necessary to search the numbers 2, 3, $4(2^2)$, 5, 7, $8(2^3)$, $9(3^2)$, 11, 13, $16(2^4)$, 17, 19, ... (not necessarily in this sequence) until the range requirement is met.

Area and time are certainly not the only two factors that affect the design cost. Sometimes several moduli requiring the same number of bits are selected such that a regular layout for each digit can be made. For example: (16,15,13,11) for a 15 bit range and (32,31,29,27,25) for a 24 bit range. On the other hand, some designers choose moduli of the forms $2^n - 1$, 2^n, $2^n + 1$ [Ta82] for simple implementation schemes and ease of binary $\Leftrightarrow$ residue conversions.

6. Conclusion

The residue arithmetic is a "concurrent" arithmetic. Two important indices "the degree of concurrency" and "the code efficiency" have been defined and analyzed. The analysis shows that the degree of concurrency in a residue number system is of order $O(N_b/\log N_b)$, where N_b is the number of bits required for binary encoding. The code efficiencies in most application ranges are between 0.91 and 0.94.

At the current state of the technology, implementations based on table look-up are preferred with respect to both performance and area for a range that can be encoded with 18 bits. If a larger range is needed, both index transform designs and array designs have smaller area requirements and delays. The array design is preferable with respect to area, and also with respect to delay if the range requires a large number of bits. However, if the array design is optimized it can be operated at a clock rate that is comparable to or less than that of the index transform approach. At feature sizes of $0.5\mu m$, table look-up designs are preferred only for ranges of up to 9 bits.

In comparing with computer arithmetic based on the binary number system it shall be noticed that our residue array multiplier is of the same complexity as a binary array multiplier. The estimated speed-up ratio is about 7 to 10 for a large application range (ranges from 2^{10} to 2^{100}). For the range greater than 2^{100} the speed-up ratio is asymptotically proportional to $N_b/\log N_b$.

Part 2. A Distributed RC Delay Line Model and MOS PLA Timing Estimation

1. Introduction

The resistance and capacitance of wires in MOS technology affect the propagation delay of signals along wires. The wire delay in state-of-the-art MOS circuits is often of the same order of magnitude as the charge transit time of transistors. The wire delay does not scale well with the technology [Me80]. Uniform scaling of circuit geometry without any change in doping levels results in constant delay for wires whose length is reduced by the scaling factor. For a wire of constant length the delay increases [Bi81]. If the wire is very long, the increase is proportional to the square of the scaling factor [Se79]. In table 1, some typical nMOS circuit parameters are listed and the effect of these parameters on the delays in a nMOS circuit is shown.

To estimate the delay in digital MOS circuits, transient analysis of a circuit that contains distributed RC lines is important. In [Pe81], a computationally simple method to estimate the delay of nodes in an RC tree network is presented. This paper develops a delay model for wires with uniformly distributed resistance and capacitance. The delay estimates are more accurate than those obtained in [Pe81] and can be applied to estimate the delay of MOS storage circuits such as PLA or ROM.

In the next section, a MOS wire is modeled as a two-port network. The distributed resistance and capacitance are approximated by lumped resistors and capacitors. A simple equation for estimating the delay of a wire with uniformly distributed resistance and capacitance is derived. The wire is assumed to be

	$\lambda = 2\mu m$		$\lambda = 0.5\mu m$	
	Resistance ($\Omega/\Box$)	Relative length of equal R	Resistance ($\Omega/\Box$)	Relative length of equal R
Transistor	10^4	1	10^4	1
Diffusion	100	100	800	12.5
Poly	25	400	200	50
Metal	0.03	3×10^5	0.24	4×10^4
	Capacitance (10^{-4} pF/μm^2)	Relative area of equal C	Capacitance (10^{-4} pF/μm^2)	Relative area of equal C
Gate	5.0	1	40	1
Diffusion	1.2	4	10	4
Poly	0.5	10	4	10
Metal	0.33	15	2.7	15

Table 1. The parasitic parameters as feature size scales down

driven by a voltage source having an internal resistance. The delay in a static nMOS PLA is estimated in section 3. The effect of reduced feature sizes is also analyzed.

2. The Distributed RC Delay Line Model

Wires in MOS circuits are often modeled as transmission lines as shown in Figure 1 [Se79], [Bi81]. Signal propagation across the wire is described by the one-dimensional diffusion equation:

$$rc\frac{\partial V(x,t)}{\partial t} = \frac{\partial^2 V(x,t)}{\partial x^2} \tag{1}$$

where $V(x,t)$ is the voltage along the wire, (t) time, (x) the spatial coordinate, and r, c the resistance and the capacitance per unit length, respectively.

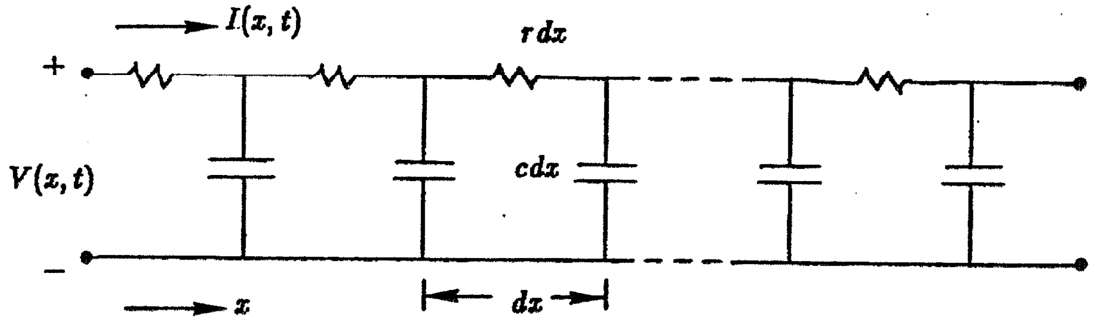


Figure 1. A wire modeled as a transmission line

The solution of this diffusion equation is complex and it is unreasonable to solve it for every element in a VLSI circuit. However, with uniformly distributed resistance and capacitance along the wire, simple approximate solutions of good accuracy can be found. In the following such a wire is referred to as a URC element [Gh68], [Pen81]. It is represented by the symbol " " and has total resistance R and total capacitance C, $R = rL$ and $C = cL$, where L is the wire length. Equation (1) is the classical equation for an inductance free transmission line with no conductance to ground. Its transmission matrix (T-matrix) in the Laplace domain is: $\begin{bmatrix} \cosh\Gamma & Z\sinh\Gamma \\ Z^{-1}\sinh\Gamma & \cosh\Gamma \end{bmatrix}$, where $\Gamma = \sqrt{sRC}$, and $Z = \sqrt{\frac{R}{sC}}$ Γ/L is often referred to as the wave propagation constant and Z as the wave impedance [De72],[Gh68].

Let $V_i(s), V_o(s), I_i(s), I_o(s)$ be the Laplace transforms of the input voltage, output voltage, input current and output current respectively, Figure 2. Then

$$\begin{bmatrix} V_i(s) \\ I_i(s) \end{bmatrix} = \begin{bmatrix} \cosh\Gamma & Z\sinh\Gamma \\ Z^{-1}\sinh\Gamma & \cosh\Gamma \end{bmatrix} \begin{bmatrix} V_o(s) \\ I_o(s) \end{bmatrix} \tag{2}$$

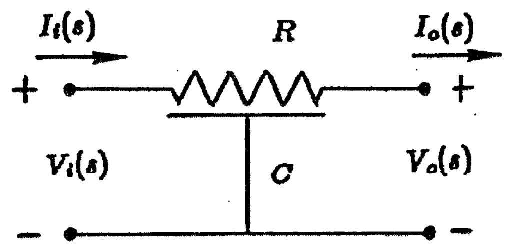
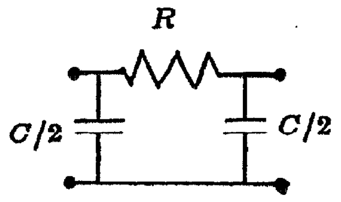


Figure 2. A URC element modeled as a two-port network

The first order Π-equivalent circuit of a URC element has a transmission impedance of R and shunt elements equal to $C/2$. This is easily realized by a Taylor series expansion of eq(2)

$$\begin{bmatrix} \cosh\Gamma & Z\sinh\Gamma \\ Z^{-1}\sinh\Gamma & \cosh\Gamma \end{bmatrix} = \begin{bmatrix} 1+0.5sRC & R \\ sC & 1+0.5sRC \end{bmatrix} + \text{high order terms} \tag{3}$$

The corresponding Π network is shown in Figure 3.



$$\text{T - Matrix} = \begin{bmatrix} 1+0.5sRC & R \\ sC & 1+0.5sRC \end{bmatrix}$$

Figure 3. A lumped approximation of a URC element

A URC Element Driven by a Voltage Source with Internal Resistance

In VLSI circuit wires are driven by transistors that often may have significant internal resistance (R_S) compared to the wire resistance (R). It is clear that the accuracy of the first order approximation increases with the ratio R_S/R. We will now study the accuracy of the first order approximation in some detail.

A URC element with total resistance R and total capacitance C driven by a voltage source having source resistance R_S is shown in Figure 4. The source resistance can be considered as the equivalent channel resistance of the driving transistor.

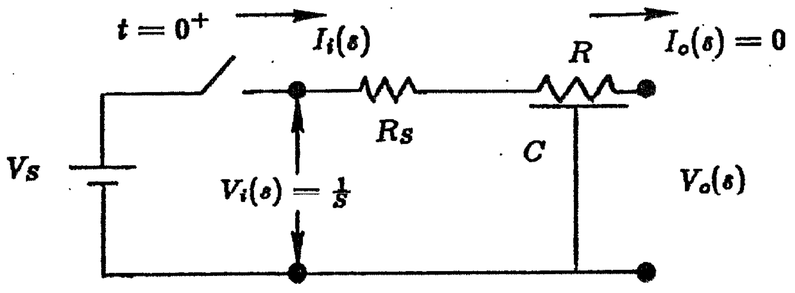


Figure 4. A URC element driven by a voltage source with source resistance R_S

The two-port network equation is

$$\begin{bmatrix} V_i(s) \\ I_i(s) \end{bmatrix} = \begin{bmatrix} 1 & R_S \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \cosh\Gamma & Z\sinh\Gamma \\ Z^{-1}\sinh\Gamma & \cosh\Gamma \end{bmatrix} \begin{bmatrix} V_o(s) \\ I_o(s) \end{bmatrix} \tag{4}$$

With open output and the input driven by a unit step voltage, *i.e.,* $I_o(s) = 0$ and $V_i(s) = 1/s$,

$$V_o(s) = \frac{1}{s(\cosh\Gamma + Z^{-1}R_S\sinh\Gamma)} \tag{5}$$

The time domain representation of the step response can be obtained by an inverse Laplace transform:

$$V_o(t) = \mathcal{L}^{-1}\left[\frac{1}{s(\cosh\Gamma + Z^{-1}R_S\sinh\Gamma)}\right] \tag{6}$$

However, there is no simple closed form for this transform. Using the first order approximation of the URC element, from (3) and (4)

$$V_o(t) = \mathcal{L}^{-1}\left[\frac{1}{s(1+s(R_S+0.5R)C)}\right]$$

This is the familiar response of a first order ordinary differential equation with time constant $\tau = (R_S + 0.5R)C$. The accuracy of this first order approximation by comparing with a numerical inverse of eq(6) is now evaluated. Some results [Im82] for different ratios of R_S/R are plotted in Figure 5. The time–axis is normalized by $(R_S + 0.5R)C$, i.e., $\alpha = t/(R_S + 0.5R)C$

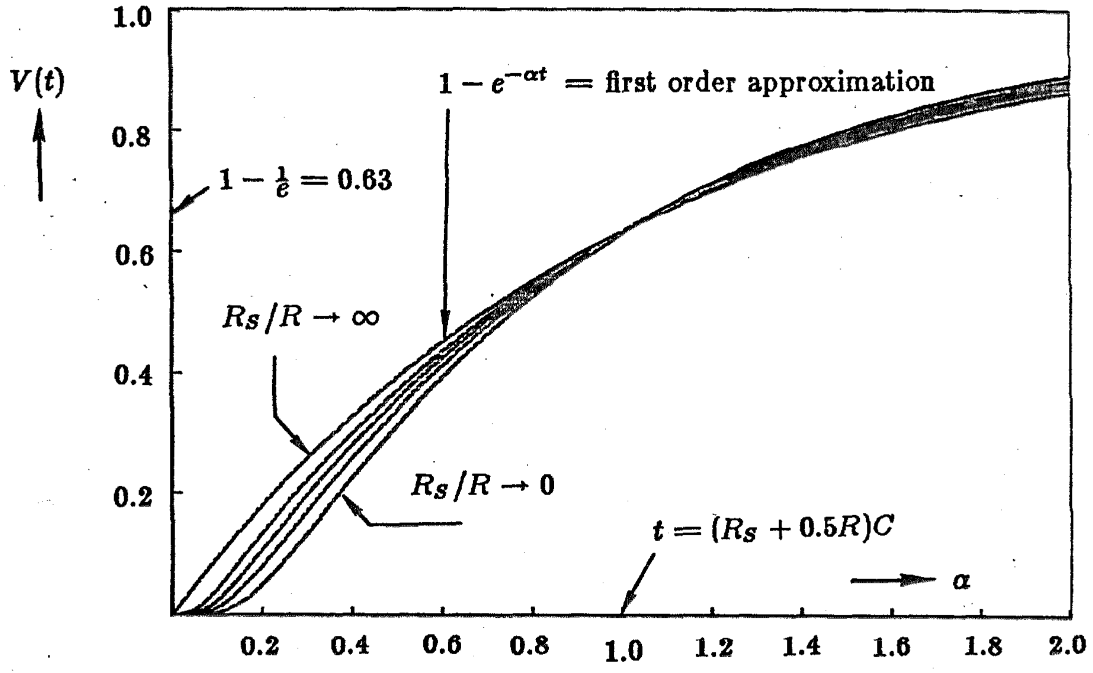


Figure 5. The unit step voltage response of the circuit in Figure 4

R_S/R	$V_o(\alpha = 1)$
∞	0.63213
10	0.63210
2	0.63180
1	0.63127
0.5	0.63044
0.1	0.62930
0	0.62922

Table 2. Unit step responses of the circuit in Figure 4

For $R = 0$, i.e., $R_S/R = \infty$, the URC element is simply a pure capacitor and the lumped model is an accurate model of the URC element. The unit step response is an exponential function with time constant $\tau = R_S C$.

For $R_S = 0$, the response is that of the URC element alone. The accuracy of the first order approximation decreases monotonically as $R_S/R \to 0$. Let $V_a(t)$ be the output voltage response of the circuit for a given ratio of R_S/R and $V_b(t)$ the response for a smaller ratio. In the beginning ($t = 0^+$), $V_b(t)$ is rising slower than $V_a(t)$ and $V_a(t) > V_b(t)$. As time proceeds, the rising rate of $V_b(t)$ is approaching the rising rate of $V_a(t)$ and eventually exceeds that rate. From Figure 5 and Table 2, it is observed that the curves for different ratios R_S/R cross at about (1, 0.63). The curves indeed intersect each other within a very small neighborhood of (1+0.01, 0.63). Hence we conclude that the time constant of the first order approximation is a very accurate approximation of the time a URC element driven by a voltage source with internal resistance reaches level $1 - \frac{1}{e}$ of its final level. For all values of R_S/R the relative error is less than 1%.

The curves in Figure 4 are bounded by the two extreme cases: $R_S/R \to \infty$ and $R_S/R \to 0$. The unit step responses of these two cases are given in (7) and (8) respectively, [Ka62].

$$V^{\infty}(t) = 1 - e^{-t/\tau} \tag{7}$$

$$V^{0}(t) = 1 + \frac{4}{\pi}\sum_{n=1}^{\infty}(-1)^{n}\frac{\exp\frac{(2n-1)^2\pi^2 t}{4\tau}}{2n-1} \tag{8}$$

While $t < 0.99\tau$ the upper bound is $V^{\infty}(t)$ and the lower bound is $V^{0}(t)$ and for $t > 1.01\tau$ the upper bound is $V^{0}(t)$ and the lower bound is $V^{\infty}(t)$. The bounds given by (7) and (8) are considerably tighter than those given in [Pe81] as is indicated in Figure 6.

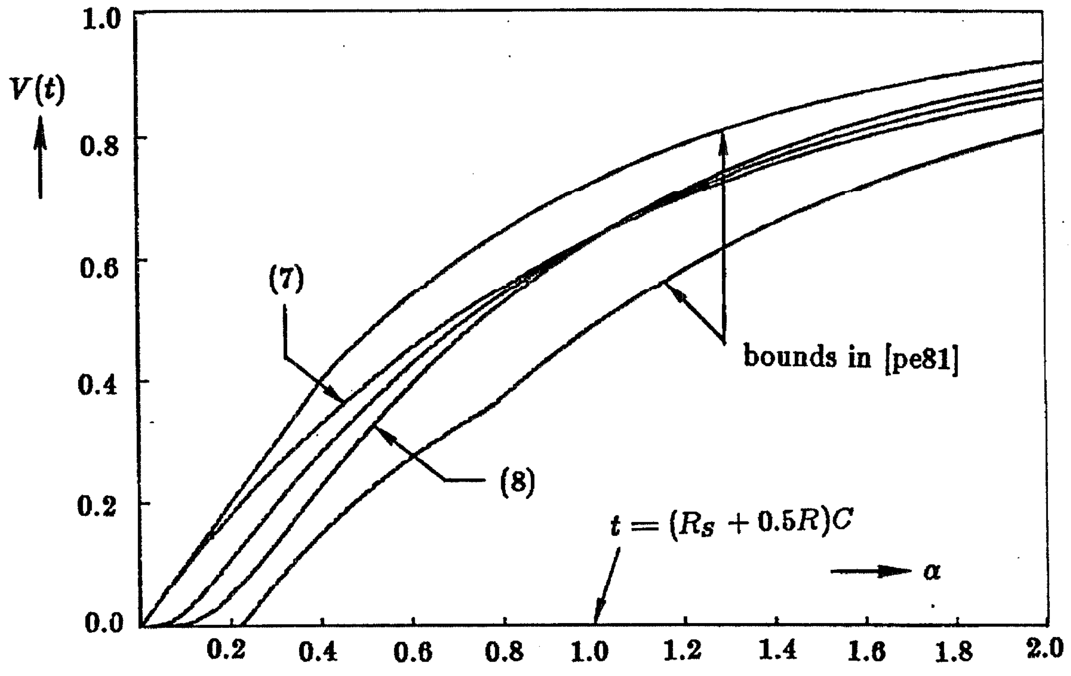


Figure 6. Bounds (7), (8) and by [Pe81] for unit step responses of the circuit in Figure 4 ($R_S/R = 1$)

A URC Element Driven by a Voltage Source with Source Resistance and Source Capacitance

A URC element with total resistance R and total capacitance C driven by a voltage source with source resistance R_S and source capacitance C_S is shown in Figure 7.

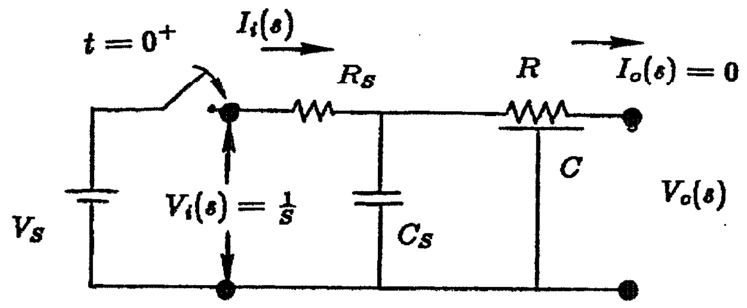


Figure 7. A URC element driven by a voltage source with source resistance R_S and source capacitance C_S

An approximation of the output voltages can again be obtained from a first order approximation of the URC element. The unit step responses depend only on two parameters: $k_r = R_S/R$ and $k_c = C_S/C$. However, over a large range of resistance and capacitance values, the unit step responses are bounded in a way similar to those shown in Figure 5 and Figure 6. Table 3 gives the unit step responses at time $t = R_S C_S + (R_S + 0.5R)C$, the time constant of the first order approximation, for different ratios k_r and k_c.

Again, we conclude that the time constant of the first order approximation is a very accurate approximation of the time constant of the circuit in Figure 7 reaches $1 - \frac{1}{e}$ of its final value.

Hence, as is the case for a one-dimensional array of lumped circuits, the time constant for a composite circuit is to first order equal to the sum of the time constants of the circuit constituting the array. Next, a delay model for a nMOS PLA will be developed.

K_c \ K_r	0.1	0.2	0.5	1	2	5	10
0.1	0.629	0.629	0.627	0.623	0.615	0.607	0.615
0.2	0.629	0.629	0.626	0.621	0.616	0.620	0.630
0.5	0.630	0.630	0.628	0.623	0.626	0.630	0.631
1.0	0.631	0.631	0.630	0.630	0.631	0.632	0.632
2.0	0.632	0.632	0.632	0.632	0.632	0.632	0.632
5.0	0.632	0.632	0.632	0.632	0.632	0.632	0.632
10	0.632	0.632	0.632	0.632	0.632	0.632	0.632

Table 3. Unit step responses of the circuit in Figure 7
at $t = R_S C_S + (R_S + 0.5R)C$

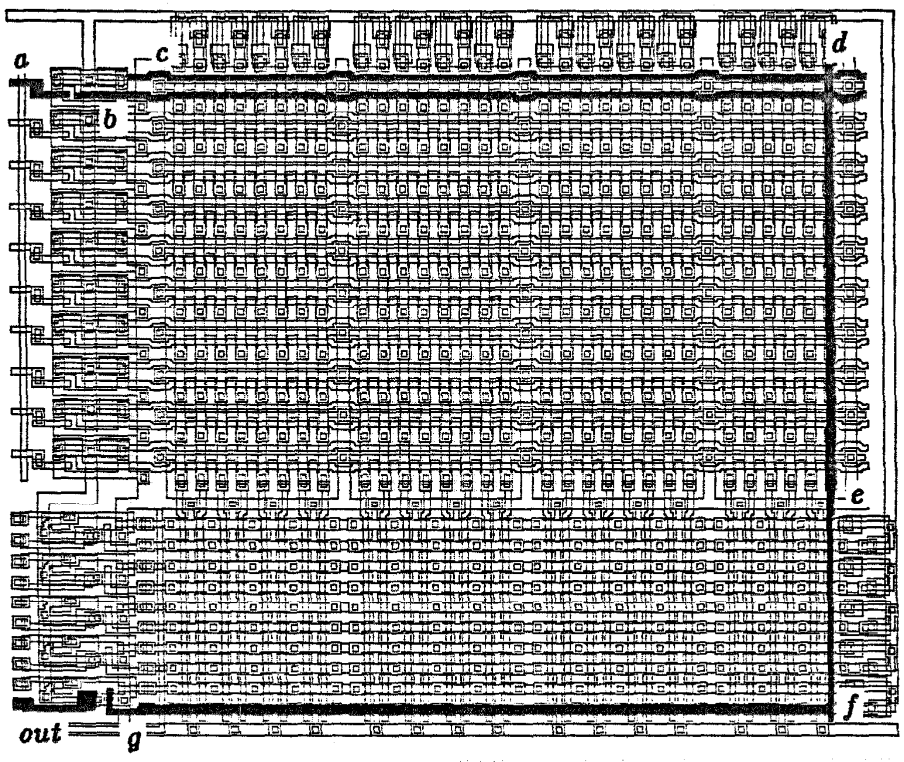
3. MOS PLA Delay Estimation

Cherry, [Ch79] has given a formula for the delay of a nMOS PLA. However, the wire delay is ignored in Cherry's model. A formula that takes the wire delay into account is developed below. The wires in the PLA are modeled as URC elements. Consider a typical nMOS PLA as shown in Figure 8. The path that passes the points a,b,c,d,f,g is a critical path. Its equivalent circuit is also shown in Figure 8.

The poly wire starting at point b that runs in parallel with the wire cd has resistance R_1 and capacitance C_1. The poly wire between point c and d has resistance R_2 and capacitance C_2. The metal wire connecting d and e has capacitance C_3. Its resistance can be ignored. The poly wire between d and f has resistance R_4 and capacitance C_4. The metal wire connecting f and g has capacitance C_5. The output loading capacitance is C_{out}. Let R_t be the channel resistance of a pull-down transistor. The pull-up to pull-down transistor ratio is denoted k, and it is assumed to be 4 in the following calculations.

Using first order approximation of the URC elements, the signal delay from the input a to the output h for $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions are given in Table 4. Wire resistances and capacitances are assumed to be uniformly distributed.

The magnitudes of $T_{0\rightarrow 1}$ and $T_{1\rightarrow 0}$ depend on the values of C_1,C_2,C_3,C_4,C_5 and C_{out}. As a rule of thumb, if the number of minterms is greater than twice the number of outputs (i.e., the AND-plane dominates the OR-plane), then $T_{0\rightarrow 1} > T_{1\rightarrow 0}$. Otherwise, $T_{1\rightarrow 0} > T_{0\rightarrow 1}$.



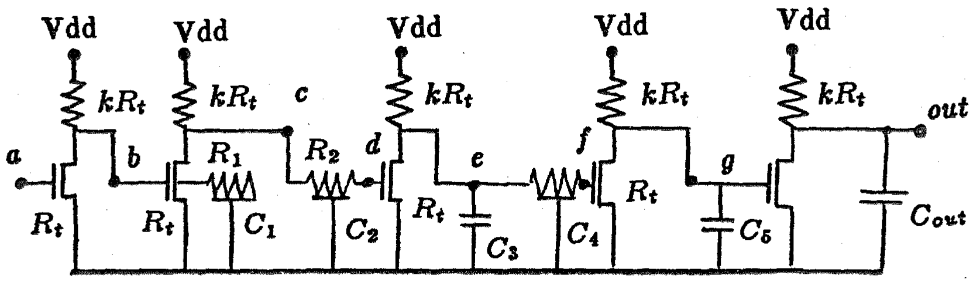


Figure 8. A critical path in a typical nMOS PLA and its equivalent circuit

Path Section	Transition $0 \to 1$	Transition $1 \to 0$
$a \to b$	$R_t C_1$	$kR_t C_1$
$b \to d$	$(kR_t + 0.5R_2)C_2$	$(R_t + 0.5R_2)C_1$
$d \to f$	$kR_t C_3 + (kR_t + 0.5R_4)C_4$	$R_t C_3 + (R_t + 0.5R_4)C_4$
$f \to g$	$kR_t C_5$	$R_t C_5$

Table 4. Delays for the PLA path shown in Figure 8

The delay from a to h for a 0 to 1 transition at point a is estimated as follows:

$$\begin{aligned} T_{0\to 1} \approx\ & R_t C_1 + kR_t C_2 && \text{—— input buffer delay} \\ +\ & 0.5R_2 C_2 && \text{—— AND-plane wire delay} \\ +\ & kR_t(C_3 + C_4) && \text{—— AND-plane switching delay} \\ +\ & 0.5R_4 C_4 && \text{—— OR-plane wire delay} \\ +\ & R_t C_5 && \text{—— OR-plane switching delay} \\ +\ & kR_t C_{out} && \text{—— output buffer delay} \end{aligned} \tag{9}$$

$T_{1\to 0}$ can be computed in a similar way. Note that $0.5R_2C_2$ and $0.5R_4C_4$ are wire delays due to the resistance of the poly wires.

Figure 9 shows the layout of a basic PLA cell. It is $7 \times 7\lambda^2$. The gates are minimum-size $(2\lambda \times 2\lambda)$ and separated by 5λ poly wires. The width is 2λ for both poly and diffusion wires. Metal wires are 3λ wide. The pull-up to pull-down transistor ratio (k) is 4.

Let r_p denote the sheet resistance of a poly wire $(\Omega/\Box)$ and C_g denote the capacitance of a minimum gate $(\text{pF}/(2\lambda)^2)$. The poly wire capacitance is about $\frac{1}{10}C_g/(2\lambda)^2$, and the metal wire capacitance is about $\frac{1}{15}C_g/(2\lambda)^2$.

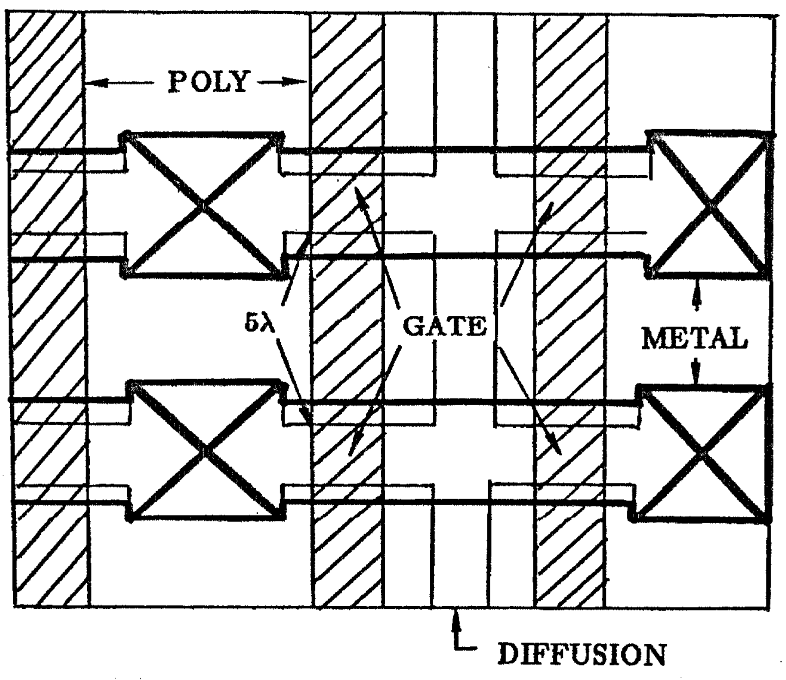


Figure 9. The layout of a basic PLA cell

The Worst Case Delay

Let P = number of minterms

I = number of inputs

O = number of outputs

$f_{out} = C_{out}/C_g$ = fan-out factor

Then the maximum value of C_i's and R_i's can be obtained under the assumption that in every cell there is a gate present.

$$\begin{aligned}
C_1 = C_2 &= \left(\left(1+\frac{5}{2}\cdot\frac{1}{10}\right)\, C_g\right) P = (1.25C_g)\, P \\
C_3 &= \left(\frac{7\times 3}{2\times 2}\cdot\frac{1}{15}C_g\right)(2I) = (0.70C_g)\, I \\
C_4 &= \left(\left(1+\frac{5}{2}\cdot\frac{1}{10}\right)\, C_g\right) O = (1.25C_g)\, O \\
C_5 &= \left(\frac{7\times 3}{2\times 2}\cdot\frac{1}{15}C_g\right) P = (0.35C_g)\, P \\
R_2 &= \left(\frac{7}{2}r_p\right) P = (3.5r_p)\, P \\
R_4 &= \left(\frac{7}{2}r_p\right) O = (3.5r_p)\, O
\end{aligned} \tag{10}$$

Substituting these values into (9) yields the longest delay,

$$T_{0\to 1} \approx \left(2.2\left(\frac{r_p}{R_t}\right)(P^2+O^2) + 6.6P + 2.8\, I + 5.0\, O + 4\, f_{out}\right)(R_t C_g) \tag{11}$$

The value of $R_t C_g$ is the time constant for a minimum size transistor driving a minimum gate. It is equal to the charge transit time ($\mathcal{T}$) of a minimum size transistor [Me80]. The values of r_p and R_t depend on the IC process technology. At 4μm feature size, R_t is of the order $10^4\Omega/\Box$ and r_p is in the range from 10 to 100 $\Omega/\Box$. Substituting r_p=25$\Omega/\Box$ and R_t=$10^4\Omega/\Box$ into (11) yields

$$T_{0\to 1} \approx \left(0.0055\,(P^2+O^2) + 6.6P + 2.8\, I + 5.0\, O + 4\, f_{out}\right)\mathcal{T} \tag{12}$$

Equation (12) gives the worst case delay of a static nMOS PLA in terms of the number of inputs I, outputs O and minterms P. For a particular design, the number of gates in each column and row is known.

Delay Estimates for Personalized PLAs

Let

$$\rho_1 = \frac{\max\{\text{the number of gates each poly wire drives in the AND plane}\}}{P}$$

$$\rho_2 = \frac{\max\{\text{the number of gates each poly wire drives in the OR plane}\}}{2 \cdot O}$$

$$\tag{13}$$

Then the relations between the actual wire capacitance (C_i') and resistance (R_i') and the maximum wire capacitance and resistance are:

$$\begin{aligned}
C_1' &\leq (1.25\rho_1 P + 0.035(1-\rho_1)P)\, C_g = (0.28 + 0.72\rho_1)C_1 \\
C_2' &\leq (1.25\rho_1 P + 0.035(1-\rho_1)P)\, C_g = (0.28 + 0.72\rho_1)C_2 \\
C_3' &= C_3 \\
C_4' &\leq (1.25\rho_2 O + 0.035(1-\rho_2)O)\, C_g = (0.28 + 0.72\rho_2)C_4 \\
C_5' &= C_5 \\
R_2' &= R_2 \\
R_4' &= R_4
\end{aligned} \tag{14}$$

where $C_1, C_2, C_3, C_4, C_5, R_2, R_4$ are the maximum wire resistances and capacitances given in (10). The approximated delay is

$$\begin{aligned}
T_{0\to 1}' \approx \Big(& 0.0055\big((0.28 + 0.72\rho_1)P^2 + (0.28 + 0.72\rho_2)O^2\big) \\
& + 6.6(0.28 + 0.72\rho_1)P + 2.8I + 5.0(0.28 + 0.72\rho_2)O + 4f_{out}\Big)\mathcal{T}
\end{aligned} \tag{15}$$

The Effect of Scaling

The coefficients in (12) and (15) are based on the resistance and capacitance values typical at the current state of the technology ($\lambda = 2\mu$m *i.e.*, r_p=25Ω/□ and R_t=10^4Ω/□). As feature sizes are reduced by α, the wire resistances/□ increase by α, while the channel resistance of a transistor essentially remains constant. The capacitances/μm^2 increase by α [Me80], [Sa82]. To summarize:

$$\begin{aligned} x &\rightarrow x/\alpha \\ R_t &\rightarrow R_t \\ C_g &\rightarrow C_g/\alpha \\ \mathcal{T} &\rightarrow \mathcal{T}/\alpha \\ r_p &\rightarrow r_p\,\alpha \\ \left(\frac{r_p}{R_t}\right) &\rightarrow \left(\frac{r_p}{R_t}\right)\alpha \end{aligned} \tag{16}$$

Equation (12) and (15) become

$$T_{0\rightarrow 1} \approx \left(0.0055\alpha(P^2+O^2) + 6.6P + 2.8\,I + 5.0\,O + 4\,f_{out}\right)\mathcal{T}' \tag{17}$$

and

$$T_{0\rightarrow 1}' \approx \Big(0.0055\alpha\big((0.28+0.72\rho_1)P^2 + (0.28+0.72\rho_2)O^2\big) + 6.6(0.28+0.72\rho_1)P + 2.8I + 5.0(0.28+0.72\rho_2)O + 4f_{out}\Big)\mathcal{T}' \tag{18}$$

where $\mathcal{T}' = \mathcal{T}/\alpha$

The wire delay is comparable with the switching delay when

$$\begin{aligned} 0.0055\alpha\, P^2 &\approx 6.6\, P & \rightarrow \quad & P \approx 1181/\alpha \\ 0.0055\alpha\, O^2 &\approx 5.0\, O & \rightarrow \quad & O \approx 909/\alpha \end{aligned} \tag{19}$$

At 0.5μm feature size $\alpha \approx 8$ and equality holds at about 150×110 bits. Hence, unless, *e.g.*, the sheet resistance is reduced, the delay of large nMOS storage structures (>10K bit) will essentially be proportional to the storage size (bits).

How to Improve the Performance

By inspecting the delay equations (17), (18), the delay introduced by a PLA can be reduced in several ways :

1. Use super buffers or a sequence of exponential horn-type drivers [Me80] to reduce the switching delay $6.6P + 2.8I + 5.0O$.
2. Reorganize the bit patterns in order to minimize $P^2 + O^2$ such that the wire delay is equally divided between the AND-plane and the OR-plane.
3. Improve the process technology to reduce r_p, and therefore the wire delay. (for example, the sheet resistance of silicide has only one tenth of that of polysilicon.)
4. Limit the driving needs on each clock phase by pipelining the operations (e.g., input buffer, AND-plane, OR-plane, output buffer).
5. Use hierarchical storage architectures as suggested in [Me80] Chapter 8.

4. Conclusion

A first order linear approximation of the one dimensional diffusion equation used to model wires in MOS technology is shown to render very accurate estimates of the time constant for wires driven by a source having internal resistance and capacitance. The first order approximation is used to derive estimates of delays in a MOS PLA. The delay is computed in terms of inputs, outputs, and minterms. The data used in section 3 is based on nMOS technology. However, the method is general and can be applied to estimate the delays of other classes of MOS storage circuits in different technologies such as CMOS or different design strategies such as precharging. The effect of scaling technologies preserving aspect ratios and doping levels, as suggested in [Me80], is calculated.

The delay equation can be used not only to estimate the speed of MOS PLA circuits but also to provide the proper VLSI computation models for storage structures.

In [Bi81], it is concluded that "the current MOS VLSI technology falls in the domain of either the synchronous or the capacitive model and for projected future technology may reach the boundary of the capacitive model region."

In that work, propagation time is defined as the time required to transfer a bit along a wire of length L. The conclusion is based on the assumption that information change is transferred on metal wires. However, for information storing and extraction, the signal is not only propagating along metal wires, but also along poly wires because of layout considerations. For a large MOS storage with flat structure (no hierarchy), the delay is essentially proportional to the storage size (i.e. the square of the chip dimension). Hence, the diffusion model is indeed the proper one in this case.

References

[Ah74] A.V. Aho, J.E. Hopcroft, J.D. Ullman, "The Design and Analysis of Computer Algorithms", 1974.

[Ba69] D. Banerji and J. A. Brzozowski, "Sign Detection in Residue Number Systems", IEEE, Transaction on Computers, Vol. C-18, Apr. 1969.

[Ba80] D. Banerji, T.Y. Cheung and V. Ganesan, "A High-Speed Division Method in Residue Arithmetic", 5th symposium on computer arithmetic. May 1980.

[Ba82] M.A. Bayoumi, G.A. Jullien and M.A. Sid-Ahmed, "VLSI implementation of Memory Intensive Residue Number System Architecture", Proceedings, 20th annual Allerton Conference on Communication, Control, and Computer. Oct 1982.

[Bi81] Jim Bilardi, M. Pracchi and F. P. Preparata, "A Critique and an Appraisal of VLSI Models of Computation", VLSI Systems and Computations, Computer Science Press, 1981, pp. 81 - 88.

[Ch79] Jim Cherry, "PLA Speed Calculations", 1979

[Ch83a] C. Chiang, "Distributed RC Delay Line Model and MOS PLA Timing Estimation", DF:5110:83, Department of Computer Science, California Institute of Technology, 1983.

[Ch83b] C. Chiang, L. Johnsson, "Residue Arithmetic and VLSI", International Conference on Computer Design (ICCD), Nov. 1983.

[Da65] L. Dadda, "Some schemes for parallel multipliers", Alta. Freq. Vol.19 May 1965.

[De72] Charles A. Desoer and Ernest S. Kuh, Basic Circuit Theory, Chapter 17, McGraw - Hill, 1969

[Et82] M. H. Etzel and W. K. Jenkins, "The Design of Specialized Residue Classes for Efficient Recursive Digital Filter Realization", IEEE, Transactions on Acoustic, Speech, and Signal Processing, Vol. ASSP-30, June 1982.

[Fo82] S. D. Fouse, G. R. Nudd and A.D Cumming, "A VLSI Architecture for Pattern Recognition Using Residue Arithmetic", IEEE, Proceeding of the 6th International Conference on Pattern Recognition, Oct 1982.

[Ga59] H. Garner, "The Residue Number System", IRE Transaction on Electronic Computers, June 1959.

[Gh68] Mohammed S. Ghausi and John J. Kelly, Introduction to Distributed-Parameter Networks: with Application to Integrated Circuits, Chapter 1,2,3, Holt,Rinehart and Winston Inc., 1968

[Ha79] G.H. Hardy and E.M. Wright, An Introduction to the Theory of Numbers, 5th edition, Oxford Science Publications, 1979.

[Hw79] K. Hwang, Computer Arithmetic, Wiley 1979.

[Im82] The IMSL Library Reference Manual, vol 2, International Mathematical &Statistical Libraries Inc., 1982.

[Je77] W. Jenkins, "The Use of Residue Number Systems in the Design of Finite Impulse Response Digital Filters", IEEE, Transaction on Circuits and Systems, Vol. CAS-24, Apr. 1977.

[Je79] W. K. Jenkins, "Recent Advances in Residue Number Techniques for Recursive Digital Filtering", IEEE, Transaction on Acoustics, Speech, and Signal Processing, Vol. ASSP-27, Feb. 1979.

[Ka62] W. M. Kaufman and S. J. Garrett, "Tapered Distributed Filters", IRE Tran. Circuit Theory, CT-9, pp. 329 - 336 Dec. 1962.

[Ki71] N.G. Kingsburg and P.J. Rayner, "Digital Filtering Using Logarithmic

Arithmetic", Electron. Lett., vol.7, Jan., 1971.

[Ki73] E. Kinoshita, H. Kosako and Y Kojima, "General Division in Symmetric Residue Number Systems", IEEE, Transaction on Computers, Vol. C-22, Feb. 1973.

[Ki82] C. Kingsley, "Earl: An Integrated Circuit Design Language" TM:4710:82, Department of Computer Science, California Institute of Technology, 1982.

[Ku78] D. J. Kuck, "The Structure of Computers and Computations", , 1978.

[Me80] Carver A. Mead and Lynn Conway, Introduction to VLSI System, Addison Wesley, 1980

[Pe81] Paul Penfield Jr. and Jorge Rubinstein, "Signal Delay in RC Tree Network", Proceedings of the Second Caltech Conference on VLSI, California Institute of Technology, 1981.

[Sa82] Krishna C. Saraswar and Farrokh Mohammadi, "Effect of Scaling of Interconnections on the Time Delay of VLSI Circuits", IEEE, Journal of Solid State Circuit, SC-17, pp. 275 -280, April 1982.

[Se79] Charles L. Seitz, "Self-Timed VLSI System", Proceedings of the First Caltech Conference on VLSI, California Institute of Technology, 1979.

[St82] S. Trimberger, "A Comparison of MOS PLAs", TM:5059:82, Department of Computer Science, California Institute of Technology, 1982.

[Sz67] N.S. Szabo and R.I. Tanaka, Residue Arithmetic and its Applications to Computer Technology, New York, McGraw-Hill, 1967.

[Ta82] F. J. Taylor, "A VLSI Residue Arithmetic Multiplier", IEEE, Transactions on Computers. Vol. C-31, June 1982.

[Ts79] B. D. Tseng and G. A. Hullien "Implementation of FFT Structures

Using the Residue Number System", IEEE, Transaction on Circuits and Systems, Vol. CAS-24, Apr. 1977.