Alternate Memory Compression Schemes for Modular Multiplication

Behrooz Parhami, *Senior Member, IEEE,* and Hsun-Feng Lai

***Abstract*—A memory compression scheme which reduces the size of the lookup tables for modular multiplication by using a new symmetry property is presented. The compression ratio for a modulus p is $(p-1)^2/\lfloor p/2 \rfloor^2$ which is approximately equal to 4 and implies a 75% savings except if p is even and small. Although this compression ratio has been achieved before, our scheme has the advantage of simpler peripheral hardware. A further benefit of our scheme is that it lends itself to additional reduction of table size by a factor of about 2 for a total savings of 87%. Realization of this additional reduction requires two stages of table lookup or more complicated addressing circuits. It is up to the system designer to decide whether this speed/cost tradeoff is worthwhile for a given application. Since throughput will be unaffected if a two-stage pipeline is used for the translation/lookup steps, this modification which achieves table compression by a factor of 8 is quite attractive in applications where long sequences of multiplications are performed. Finally, we show that by using a multiplication algorithm based on squaring, a compression ratio of $(p-1)^2/(2p-1)$ or roughly $p/2$ is achievable with moderate hardware complexity and two lookup steps.**

I. Introduction

Modular arithmetic has been used in digital filtering and other signal processing applications for years [6]. Operations like $(x + y) \bmod p$ or $(x \times y) \bmod p$, where x and y belong to the residue class $Z_p = \{0, 1, 2, \cdots, p-1\}$ are common in modular arithmetic. Since the computation of modulo-p residues is complicated for an arbitrary p and this may increase the hardware complexity and operation latencies significantly, it is generally advocated that table lookup methods be used for modular arithmetic. By using table lookup, we are no longer limited to moduli of form $2^k - 1$, as suggested in [5], in order to simplify the computation of residues.

Unlike weighted number systems, for which rounding is applicable, residue arithmetic deals with exact results and thus suffers from the problem of unrecoverable overflow. There are two ways to solve this problem. One is to introduce approximation errors which entails overhead. The other is to enlarge the dynamic range of the residue number system (RNS). Let $P = \{p_1, \cdots, p_L\}$ be the pairwise relatively prime set of moduli in an RNS. The dynamic range of this RNS is taken to be $[-V/2, V/2)$, where $V = \Pi_{1 \le i \le L} p_i$. Either more moduli or larger moduli are required in the latter approach. Both methods will result in additional hardware, with the delay increasing as well if larger moduli are employed.

Manuscript received October 16, 1990; revised April 6, 1992.

B. Parhami is with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106.

H.-F. Lai is with Epic Design Technology Inc., Santa Clara, CA.

IEEE Log Number 9206038.

A modular multiplication table is composed of entries $T_p[i, j] = (i \times j) \bmod p$ where $i, j \in Z_p$. It can be shown that T_p possesses diagonal and antidiagonal symmetry. This symmetry property is used to construct two tables of sizes $\lfloor p/2 \rfloor$ and $\lfloor p/2 \rfloor^2$ in [1]. The main disadvantages of this scheme are that two separate tables are needed and that the addressing protocol is complicated. The compression ratio for the above proposed scheme is about 4, leading to a 75% savings.

In this paper, we first exploit a different symmetry property that can reduce the size of the lookup table by the same factor of 4 but results in a simpler mechanism for addressing. Only one table is needed for each modulus and the peripheral hardware is less complex. A further compression based on the commutativity of multiplication yields an overall reduction factor of 8. However, since the final table resulting from this scheme is triangular, indirect addressing or more complicated address translation must be used for each access. Finally, we adapt a multiplication algorithm based on squaring to modular arithmetic and study the resulting compression ratio and hardware complexity.

II. The Complement-Symmetry Property

Fig. 1 shows the modular multiplication table T_p with $T_p[i, j] = (i \times j) \bmod p$. Since the multiplication result with one zero operand is zero, we can use a smaller table by detecting these special cases and bypassing the table access. In the following discussion, T_p refers to the modular multiplication table with entries for zero operands removed. We will see later that the removal of zero entries can in fact complicate the addressing protocol, nullifying the savings in size. Nevertheless, we proceed with this assumption to make our results directly comparable to those of [1].

The following lemma states the complement-symmetry property of T_p.

Lemma 1: For any modular multiplication table T_p, we have

$$T_p[p-x, p-y] = T_p[x, y] \quad (1)$$

$$T_p[x, p-y] = T_p[p-x, y] = p - T_p[x, y]. \quad (2)$$

$y \backslash x$	0	1	2	3	...	$\lfloor p/2 \rfloor$	...	$p-2$	$p-1$
0	0	0	0	0	...	0	...	0	0
1	0	1	2	3	...	$\lfloor p/2 \rfloor$	...	$p-2$	$p-1$
2	0	2							...
3	0	3							...
...	...	...							...
$\lfloor p/2 \rfloor$	0	$\lfloor p/2 \rfloor$							...
...	...	...							...
$p-2$	0	$p-2$							2
$p-1$	0	$p-1$ ...	...	...	...	...	...	2	1

Fig. 1. The complete modulo-p multiplication table.

T_p

$y \backslash x$	1	2	3	...	$\lfloor p/2 \rfloor$	...	$p-2$	$p-1$
1	1	2	3	...	$\lfloor p/2 \rfloor$	...	$p-2$	$p-1$
2	2							...
3	3							...
...	...							...
$\lfloor p/2 \rfloor$	$\lfloor p/2 \rfloor$							...
...	...							...
$p-2$	$p-2$							2
$p-1$	$p-1$ ...	...	...	...	...	...	2	1

T_p'

$y' \backslash x'$	1	2	3	...	$\lfloor p/2 \rfloor$
1	1	2	3	...	$\lfloor p/2 \rfloor$
2	2				...
3	3				...
...	...				...
$\lfloor p/2 \rfloor$	$\lfloor p/2 \rfloor$				...

Fig. 2. Reduction of the modular multiplication table T_p.

Proof: We prove (1) as follows:

$$\begin{aligned} T_p[p-x, p-y] &= [(p-x) \times (p-y)] \bmod p \\ &= (p^2 - px - py + xy) \bmod p \\ &= (xy) \bmod p \\ &= T_p[x, y]. \end{aligned}$$

To prove (2), we begin by writing

$$\begin{aligned} T_p[x, p-y] &= [x \times (p-y)] \bmod p \\ &= (px - xy) \bmod p \\ &= (-xy) \bmod p \\ &= p - [(xy) \bmod p] \\ &= p - T_p[x, y]. \end{aligned}$$

The second part of (2) can be similarly established. ■

Theorem 1: A modular multiplication table T_p for the modulus p can be reduced to the smaller table T_p' with $|T_p|/|T_p'| = 4 - \epsilon_p$, where $|T|$ is the size of T and $\epsilon_p = 8/p - 4/p^2$ for even p, $\epsilon_p = 0$ for odd p.

Proof: By Lemma 1, the elements $T_p[p-x, p-y]$, $T_p[x, p-y]$, and $T_p[p-x, y]$, can be obtained from $T_p[x, y]$. Thus T_p can be reduced to its upper left quadrant T_p' which is of size $\lfloor p/2 \rfloor$ by $\lfloor p/2 \rfloor$ as shown in Fig. 2. The table size compression ratio then is $|T_p|/|T_p'| = (p-1)^2/\lfloor p/2 \rfloor^2 = 4 - \epsilon_p$. If p is odd, then $\lfloor p/2 \rfloor = (p-1)/2$ and $\epsilon_p = 0$ follows. For even values of p, the compression ratio is $|T_p|/|T_p'| = 4(p-1)^2/p^2 = 4 - 8/p + 4/p^2$, leading to the desired result. ■

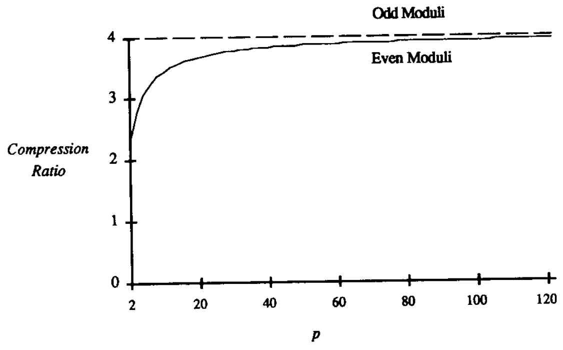


Fig. 3. Table size compression ratio as a function of p using complement-symmetry.

Fig. 3 shows the variation of the table size compression ratio as function of p. Any RNS has at most one even modulus. As a direct consequence of Theorem 1, one should try to make the even modulus as large as possible to achieve better compression. In practice, this is done anyway for performance reasons [7] since by making the even modulus comparable to the largest odd modulus utilized (combining it with some odd modulus or picking a suitably large power of 2), the range can be expanded with no speed penalty.

Let us consider an example. The modular multiplication tables T_7 and T_8 are shown in Fig. 4, with four related entries circled in each table for illustration. Fig. 5 shows the tables after reduction by Theorem 1. Variables x' and

T_7

$y \backslash x$	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

T_8

$y \backslash x$	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7
2	2	4	6	0	2	4	6
3	3	6	1	4	7	2	5
4	4	0	4	0	4	0	4
5	5	2	7	4	1	6	3
6	6	4	2	0	6	4	2
7	7	6	5	4	3	2	1

Fig. 4. Modular multiplication tables for $p = 7$ and $p = 8$.

T_7'

$y' \backslash x'$	1	2	3
1	1	2	3
2	2	4	6
3	3	6	2

T_8'

$y' \backslash x'$	1	2	3	4
1	1	2	3	4
2	2	4	6	0
3	3	6	1	4
4	4	0	4	0

Fig. 5. Reduced modular multiplication tables T_7' and T_8'.

y' in Fig. 5 are new indices used to access the reduced tables. In general, we have $1 \leq x', y' \leq \lfloor p/2 \rfloor$.

A. The Addressing Protocol

The addressing protocol transforms the input operands x and y into the indices x' and y' which will be used for addressing the reduced table T_p'. This procedure can be described by the following functions:

$$\begin{aligned} x' &= x && \text{if } x \leq \lfloor p/2 \rfloor \\ &= p - x && \text{if } x > \lfloor p/2 \rfloor \end{aligned} \tag{3}$$

$$\begin{aligned} y' &= y && \text{if } y \leq \lfloor p/2 \rfloor \\ &= p - y && \text{if } y > \lfloor p/2 \rfloor . \end{aligned} \tag{4}$$

So after the transformation, x' and y' will fall in the range $[1, \lfloor p/2 \rfloor]$. From (2), if we do only one complementation, we have to complement the returned data to obtain the correct result. A flag c, which is initialized to "false" and is complemented with each operand complementation, can be introduced to keep track of this information. If we need no complementation or if we complement both operands, c will be false and the result fetched from the table will be correct. If c is true, we need to complement the fetched table entry before returning it.

B. Implementation Issues

Off-the-shelf read-only memories (ROM's) come in standard sizes of the form $2^{2k} \times 2^w$. For example take the case of $p = 23$. Without compression, the required ROM size is 484×5, assuming zero suppression. In a custom-built circuit, a ROM of this size may be employed but then addressing becomes nontrivial since the address expression $22(y - 1) + x - 1$ requires a multiplication. It is more likely that a ROM of size 1024×5 is used for ease of addressing so that x and y each determine 5 b of the 10-b address. With an off-the-shelf ROM, the size is likely to be 1024×8. In the latter case, error checking in the form of parity bits can be incorporated into the design with little extra cost. After our reduction, the required size will be 121×5, with practical sizes being 256×5 and 256×8 to avoid the address computation $11(y' - 1) + x' - 1$. It is easy to see that the actual compression is by a factor of four in each case. The same observations apply to even moduli.

Zero suppression may cause problems from a practical point of view since $x' - 1$ and $y' - 1$ must be used as indices into the reduced table. These extra computations can be avoided by including the zero entries in the table. Since as discussed above, in most cases the table is larger than required, we can include entries corresponding to zero operands with no table size penalty. This can lead to substantial savings in the cost and delay of the peripheral circuits for table access. For borderline cases, such as $p = 33$, removing the zeros can lead to substantial reduction in table size. In such cases, it is up to the designer to decide whether the complexity and delay added by the zero handling hardware and operand modification circuits are worthwhile.

It is worth noting that a two-level table lookup can be utilized to effect address computations resulting from nonstandard table sizes. The multiplicative part of an address computation such as $11(y' - 1) + x' - 1$ or more generally $\lfloor p/2 \rfloor (y' - 1) + x' - 1$ can be handled by a lookup table B_p' as shown in Fig. 6 and the two parts of the computation pipelined to achieve the same throughput as before. Fig. 7 contains two examples for the B_p' and T_p' tables. The tables B_p' and T_p' can be merged into one table to reduce the implementation cost, but this will affect both the speed (sequential rather than cascaded operation) and the throughput (loss of pipelining capability). An offset equal to the size of B_p' should be added to the original entries of B_p' before B_p' and T_p' are merged. Figs. 8 and 9 depict the general scheme and examples of merged tables.

C. Comparison with Prior Work

The reduction scheme reported in [1] is based on the diagonal and antidiagonal symmetries of T_p. Since after removing the redundancies, the remaining one-fourth of the elements do not form a rectangular subarray, their addressing protocol is somewhat more complicated than ours. It starts by computing x' and y' as in (3) and (4). However, they need to treat the case $x' = y'$ by including a separate one-dimensional table. Furthermore, for the case $x' \neq y'$, the larger of x' and y' must be determined and assigned to the table access index a or b according to the value of the flag c. The other index then assumes the smaller of the two values and $T[a, b]$ is fetched.

Our scheme removes the need for a separate "diagonal" table and replaces the comparison and assignment

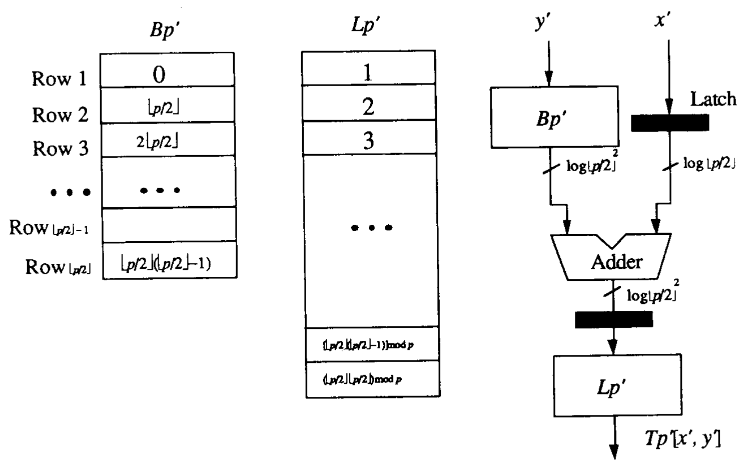
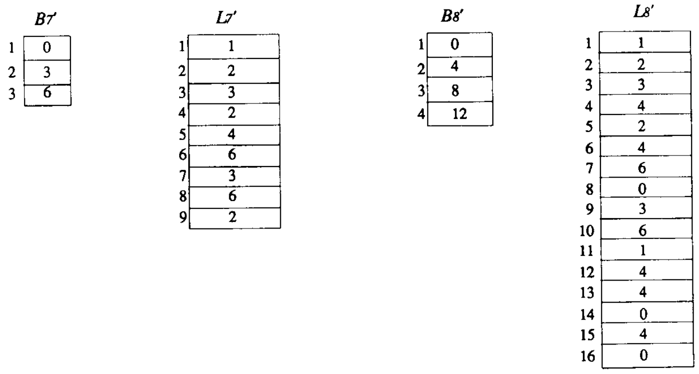


Fig. 6. Pipelined indirect addressing implementation of T'_p.



	$B7'$
1	0
2	3
3	6

	$L7'$
1	1
2	2
3	3
4	2
5	4
6	6
7	3
8	6
9	2

	$B8'$
1	0
2	4
3	8
4	12

	$L8'$
1	1
2	2
3	3
4	4
5	2
6	4
7	6
8	0
9	3
10	6
11	1
12	4
13	4
14	0
15	4
16	0

Fig. 7. Table contents for B'_7, L'_7, B'_8, and L'_8.

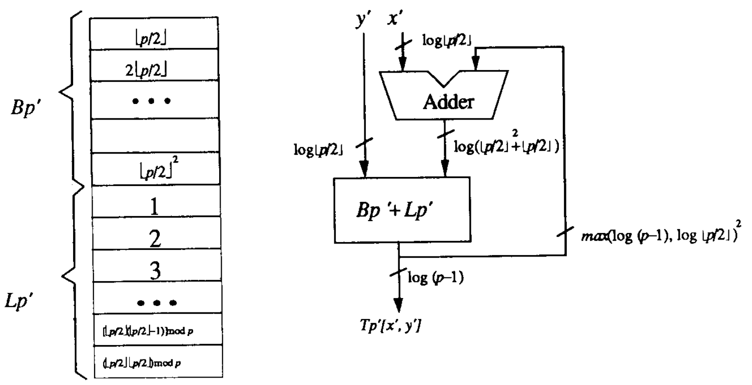


Fig. 8. Merged indirect addressing implementation of T'_p.

steps outlined above by a selective complementation step for the fetched value. Even if this complementation is done through a full-fledged subtraction, our scheme will still be faster. Furthermore, when our scheme is used with an asynchronous mode of operation, the complementation time can be saved in 50% of the cases when the fetched entry is the correct result. No such saving is possible with the previous scheme.

III. The Commutativity Property

Inspection of the reduced tables in Fig. 5 reveals a diagonal symmetry property of T'_p that can be used to further reduce their sizes. This symmetry is due to the commutativity of modular multiplication; i.e., $(x \times y) \bmod p = (y \times x) \bmod p$ since $x \times y = y \times x$. Fig. 10 shows the parts of these example tables that need to be stored.

	$B7' + L7'$
1	3
2	6
3	9
4	1
5	2
6	3
7	2
8	4
9	6
10	3
11	6
12	2

	$B8' + L8'$
1	4
2	8
3	12
4	16
5	1
6	2
7	3
8	4
9	2
10	4
11	6
12	0
13	3
14	6
15	1
16	4
17	4
18	0
19	4
20	0

Fig. 9. The tables B_7', L_7' and B_8', L_8' after merging.

$T7''$

$y'' \backslash x''$	1	2	3
1	1	2	3
2		4	6
3			2

$T8''$

$y'' \backslash x''$	1	2	3	4
1	1	2	3	4
2		4	6	0
3			1	4
4				0

Fig. 10. Further reduction for modular multiplication tables T_7'' and T_8''.

Theorem 2: A reduced modular multiplication table T_p' for the modulus p can be further transformed to the smaller table T_p'' with $|T_p'|/|T_p''| = 2 - \epsilon_p'$, where $|T|$ is the size of T and $\epsilon_p' = 2/(\lfloor p/2 \rfloor + 1)$.

Proof: By commutativity of multiplication, we can remove all $T_p'[x', y']$ entries for which $x' > y'$. The remaining upper right triangle has $\lfloor p/2 \rfloor (\lfloor p/2 \rfloor + 1)/2$ entries. Thus $|T_p'|/|T_p''| = 2\lfloor p/2 \rfloor^2 / [\lfloor p/2 \rfloor (\lfloor p/2 \rfloor + 1)] = 2 - 2/(\lfloor p/2 \rfloor + 1) = 2 - \epsilon_p'$. ■

Corollary 1: The overall compression ratio achieved by applying the reductions defined by Theorems 1 and 2 is $|T_p|/|T_p''| = 8 - \epsilon_p''$, where $\epsilon_p'' = (32p - 8)/(p^2 + 2p)$ for even p, $\epsilon_p'' = 16/(p + 1)$ for odd p.

Proof: From Theorems 1 and 2, we can write $|T_p|/|T_p''| = (|T_p|/|T_p'|) \times (|T_p'|/|T_p''|) = [(p - 1)^2/\lfloor p/2 \rfloor^2] \times 2\lfloor p/2 \rfloor^2/[\lfloor p/2 \rfloor (\lfloor p/2 \rfloor + 1)] = 8 - \epsilon_p''$. The value of ϵ_p'' can be obtained for even and odd values of p as in the proof of Theorem 1. ■

Fig. 11 depicts the variation of the table size compression ratio as a function of p. Again, odd moduli enjoy higher compression ratios.

A. The Addressing Protocol

To assure that indices for accessing T_p'' fall in the proper range, x' and y' have to be transformed into x'' and y'' through a simple comparison. This transformation can be

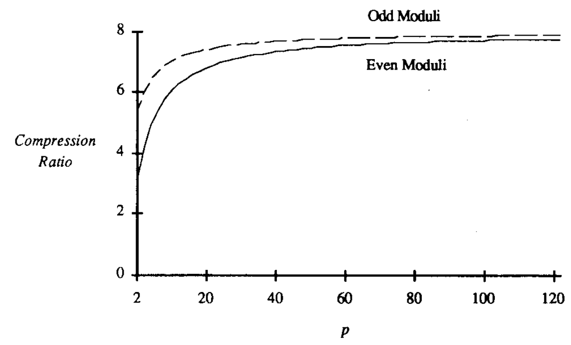


Fig. 11. Table size compression ratio as a function of p using commutativity.

	$B7''$
1	0
2	3
3	5

	$L7''$
1	1
2	2
3	3
4	4
5	6
6	2

	$B8''$
1	0
2	4
3	7
4	9

	$L8''$
1	1
2	2
3	3
4	4
5	4
6	6
7	0
8	1
9	4
10	0

Fig. 12. Table contents for B_7'', L_7'' and B_8'', L_8''.

described by the following functions:

$$x'' = \min(x', y') \tag{5}$$

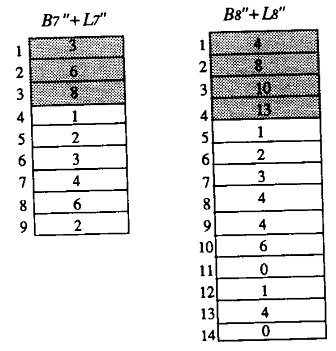
$$y'' = \max(x', y') \tag{6}$$

Thus assuming that a triangular table can be stored directly, the element $T_p''[x'', y'']$ will provide the desired result.

B. Implementation Issues

The main implementation problem for T_p'' arises from its triangular shape. A straightforward approach for using this reduction is to pack two triangular tables into a square ROM. This will reduce the speed roughly by a factor of 2 since the two tables cannot be accessed simultaneously. Throughput reduction may be somewhat less severe if the merged tables are associated with separate address computation circuits and pipelining is used. Another approach would be to store the table in row-major order and apply the two-level addressing scheme (two sequential table accesses) discussed at the end of Section II-B to effect the address computation $\lfloor p/2 \rfloor x'' - x''(x'' - 1)/2 + y'' = f_p(x'') + y''$ through a lookup table for f_p and an adder. Fig. 12 shows examples of the tables B_p'' and T_p'' that are required for the indirect addressing scheme and Fig. 13 shows the merged versions of the tables.

In the remainder of this subsection, we explore the possibility of rearranging some of the entries to convert the triangular table into a rectangular one and deal with the associated address translation schemes. Consider an $n \times$



	$B_7''+L_7''$
1	3
2	6
3	8
4	1
5	2
6	3
7	4
8	6
9	2

	$B_8''+L_8''$
1	4
2	8
3	10
4	13
5	1
6	2
7	3
8	4
9	4
10	6
11	0
12	1
13	4
14	0

Fig. 13. Merged table contents for $B_7'' + L_7''$ and $B_8'' + L_8''$.

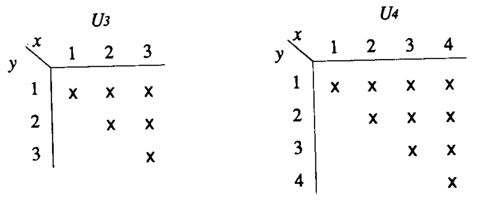


Fig. 14. Triangular tables U_3 and U_4.

n upper triangular table U_n as in the examples of Fig. 14 (lower triangular tables can be handled similarly and thus will not be discussed here). We can move some of the elements from the lower right corner to the unused slots in the upper left corner. Fig. 15 shows the process of converting U_3 and U_4 into 2×3 and 3×4 rectangular tables. The resulting R_3 and R_4 tables are depicted in Fig. 16 ("x" denotes an original table entry, "O" represents a displaced entry, and "b" is a blank or unused entry). The general rearrangement process is shown in Figs. 17 through 20. Conceptually, the indices x and y of $U_n[x, y]$ are transformed by the following steps:

1) Change x to $x' = n + 1 - x$ if $y > \lfloor n/2 \rfloor + 1$ (see Fig. 17).

2) Change y to $y' = n + 1 - y$ if $y > \lfloor n/2 \rfloor + 1$ (see Fig. 18).

3) Change y' to $y' + 1$ for elements transformed by the above rules (see Fig. 19).

The final rectangular table is shown in Fig. 20. Thus a rearranged entry $U_n[x, y]$ moves to the location $R_n[n + 1 - x, n + 2 - y]$ in the corresponding rectangular table. The above arguments lead to the following result.

Theorem 3: For any upper $n \times n$ triangular table U_n, we can form a rectangular table R_n of size $(\lfloor n/2 \rfloor + 1) \times n$ such that the original table entry $U_n[x, y]$ is either at location $R_n[x, y]$ or at location $R_n[n + 1 - x, n + 2 - y]$.

Proof: The number of rows in the rearranged rectangular table will be $\lfloor n/2 \rfloor + 1$. Since each $y > \lfloor n/2 \rfloor + 1$ is changed to $n + 2 - y$, which is at most $n - \lfloor n/2 \rfloor \leq \lfloor n/2 \rfloor + 1$, each rearranged entry will have a row index of at most $\lfloor n/2 \rfloor + 1$. What remains to be

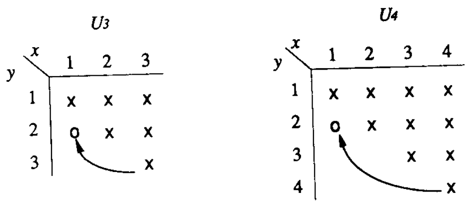


Fig. 15. Moving the lower right elements to the upper left slots.

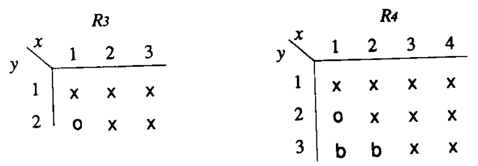


Fig. 16. The resulting rectangular tables.

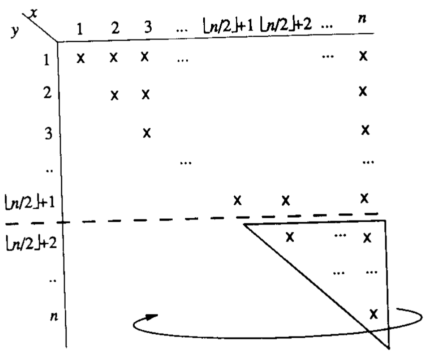


Fig. 17. The first step in the conceptual address transformation.

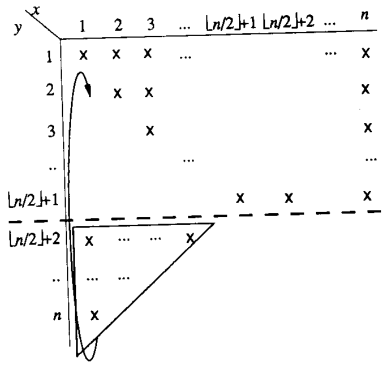


Fig. 18. The second step in the conceptual address transformation.

established is that all rearranged entries actually fall in the empty slots. For this to happen, we must have $n + 1 - x < n + 2 - y$ or $x \geq y$, which is true of any rearranged entry $U_n[x, y]$. ■

C. Comparison with Prior Work

The scheme just described allows a further table size reduction by a factor of about 2 at the expense of more

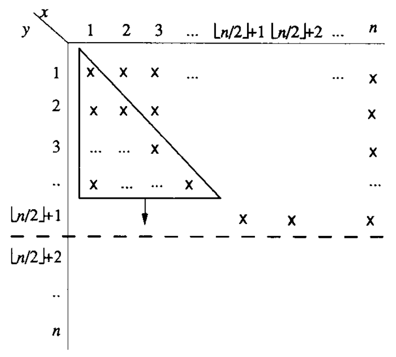


Fig. 19. The third step in the conceptual address transformation.

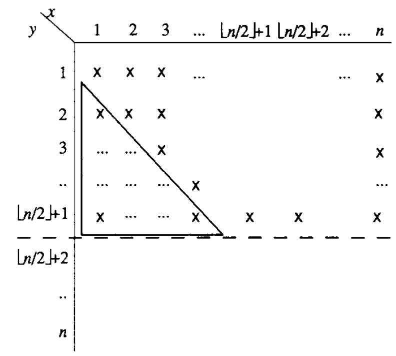


Fig. 20. The final rectangular table.

complicated peripheral circuits. This reduction scheme is probably worthwhile only in situations where pipelining can be used to offset its negative effect on throughput. If a two-level table lookup arrangement is used, the throughput will be comparable to our initial scheme (compression ratio of about 4) and to the scheme proposed in [1]. With address translation to obtain a rectangular table, the multiplication pipeline will include an extra stage and pipeline drainage [4] due to data dependencies and conditional branches will become more likely. Thus the latter scheme must be used with care.

IV. Multiplication by Squaring

In a recent paper, Ling [3] describes a table compression scheme for ordinary (not modular) multiplication through squaring. He takes advantage of the equality

$$xy = [(x+y)^2 - (x-y)^2]/4 \tag{7}$$

and suggests that $n \times n$ multiplication can be performed by two accesses to a table of size 2^{n+1} rather than through a lookup table of size 2^{2n} (actually through minor adjustments to the above, he arrives at a table size of 2^n, but this simpler view is sufficient for our purposes). We now show that the same scheme is applicable to modular multiplication. We first prove the following result for modular multiplication.

Theorem 4:

$$(xy) \bmod p = [\lfloor (x+y)^2/4 \rfloor \bmod p - \lfloor (x-y)^2/4 \rfloor \bmod p] \bmod p. \tag{8}$$

Proof: We note that $x + y$ and $x - y$ are either both even or both odd. If they are even, then (8) is equivalent to (7). If they are odd, then $\lfloor (x+y)^2/4 \rfloor$ is 1/4 of a unit less than $(x+y)^2/4$ and $\lfloor (x-y)^2/4 \rfloor$ is 1/4 of a unit less than $(x-y)^2/4$. Thus in either case we can write

$$[(x+y)^2 - (x-y)^2]/4 = \lfloor (x+y)^2/4 \rfloor - \lfloor (x-y)^2/4 \rfloor .$$

This concludes the proof. ■

A. The Addressing Protocol

The addressing protocol consists of computing $z' = x + y$ and $z'' = x - y$ and then accessing the $f(z) = \lfloor z^2/4 \rfloor \bmod p$ table twice to obtain $\lfloor (x+y)^2/4 \rfloor \bmod p$ and $\lfloor (x-y)^2/4 \rfloor \bmod p$. A final modulo-$p$ subtraction provides the desired result. Since $x + y$ is in the range $[0, 2p - 2]$, the table size will be $2p - 1$.

Although it is possible to use $\lfloor (x+y)/2 \rfloor$ and $\lfloor (x-y)/2 \rfloor$ instead of z' and z'' in order to reduce the table size by about 50% in a manner similar to Ling's multiplication scheme [3], it is our judgment that the additional peripheral circuits cannot be justified in view of the fact that $2p - 1$ is already quite small in practice.

B. Implementation Issues

The two table accesses required can be performed sequentially or in parallel. With sequential accesses and pipelining, four main steps are required:

1) Compute $x + y$.
2) Compute $|x - y|$ and lookup $\lfloor (x+y)^2/4 \rfloor \bmod p$.
3) Lookup $\lfloor (x-y)^2/4 \rfloor \bmod p$.
4) Subtract, modulo p, the above two entries.

The next multiplication can be initiated once the current one is in step 3. Thus, we can have pipelining within a single multiplication (pipelining period of 1 cycle) and between successive multiplications (pipelining period of 2 cycles).

Parallel tables can improve both the delay and the pipelining period for successive multiplications. The steps in this case are:

1) Compute $x + y$ and $|x - y|$.
2) Lookup $\lfloor (x+y)^2/4 \rfloor \bmod p$ and $\lfloor (x-y)^2/4 \rfloor \bmod p$.
3) Subtract, modulo p, the above two entries.

The delay has been reduced to 3 cycles and a new multiplication can be initiated in each cycle. This gain is achieved by the extra cost of another table and the associated input adder.

C. Comparison with Other Schemes

The table size reduction for this scheme is from $(p - 1)^2$ to $2p - 1$ with sequential access and to $4p - 2$ with parallel tables and supporting hardware. However, since residue arithmetic based on table lookup requires fairly small moduli, the previous two schemes remain competitive due to their simpler peripheral hardware. As a specific example, to achieve a range of about 48 binary digits, moduli set $\{7, 11, 13, 17, 19, 23, 25, 27, 29, 31, 32\}$ is adequate. The largest modulus in this set requires a table with 1024 words before compression. Our first compression scheme yields a table of size 256. Our extended compression scheme reduces the table size to 128 words. The squaring approach needs two 64-word tables that are accessed in parallel if it is to be competitive with respect to speed.

V. Conclusion

We have presented three new compression schemes for modular multiplication tables. Our first scheme provides the same compression ratio as that offered in [1], but has a simpler addressing protocol and a more uniform memory structure.

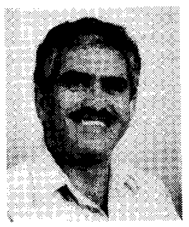
We have also shown that our new scheme lends itself to a further table size reduction by a factor of 2 with more complicated peripheral hardware, thus presenting the designer with an opportunity for speed/cost tradeoff. Because the additional hardware is used in an extra stage between the original address translator and the lookup table (without affecting the logical structure or the complexity of either of these two components), achieving the same throughput is quite easy with a pipelined organization. The scheme is thus quite attractive in applications where long sequences of multiplications are performed.

Additionally, we have discussed a compression scheme based on squaring that is particularly attractive for large moduli since it reduces the table size from $(p - 1)^2$ to $2p - 1$ with the slower sequential table access and to $4p - 2$ for the higher speed parallel version.

In practice the various parameters of the three table compression schemes proposed in this paper are fairly close and the optimal selection will depend on speed/cost requirements as well as the parameters of the technology used for implementation.

References

[1] C. H. Huang and F. J. Taylor, "A memory compression scheme for modular arithmetic," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-27, pp. 608-611, Dec. 1979.
[2] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*. New York: McGraw-Hill, 1984.
[3] H. Ling, "An approach to implementing multiplication with small tables," *IEEE Trans. Comput.*, vol. 39, no. 5, pp. 717-718, May 1990.
[4] P. M. Kogge, *The Architecture of Pipelined Computers*. New York: McGraw-Hill, 1981.
[5] R. D. Merill, Jr., "Improved digital computer performance using residue number theory," *IEEE Trans. Electron. Comput.*, vol. EC-13, no. 2, pp. 93-101, Apr. 1964.
[6] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Taylor, *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*. New York: IEEE Press, 1986.
[7] S. Waser and M. J. Flynn, *Introduction to Arithmetic for Digital Systems Designers*. Holt, Rinehart and Winston, 1982.



Behrooz Parhami (S'70-M'73-SM'78) received the Ph.D. degree in computer science from the University of California, Los Angeles, in 1973.

Presently, he is Professor in the Department of Electrical and Computer Engineering, University of California, Santa Barbara. His current research deals with parallel architectures and algorithms, high-speed computer arithmetic, and dependable (fault-tolerant) computing. In his previous position with Sharif University of Technology in Tehran, Iran (1974-1988), he was also involved in the areas of educational planning, curriculum development, standardization efforts, technology transfer, and had various editorial responsibilities, including a five-year term as Editor of *Computer Report*, a Farsi-language computing periodical. His technical publications include over 90 papers in journals and international conferences, two Farsi-language textbooks, and an English/Farsi glossary of computing terms.

Dr. Parhami is a member of the Association for Computing Machinery and the British Computer Society and a Distinguished Member of the Informatics Society of Iran for which he served as a founding member and President during 1979-1984. He also served as Chairman of IEEE Iran Section (1977-1986) and received the IEEE Centennial Medal in 1984.

Hsun-Feng Lai was born in Taipei, Taiwan, Republic of China, on May 4, 1964. He received the B.S. degree in computer science and information engineering from the National Taiwan University in 1986 and the M.S. degree in computer engineering from the University of California, Santa Barbara, in 1991.

Since August 1991, he has been with Epic Design Technology Inc., where his responsibilities include transistor-level timing and power simulation. His areas of technical and research interest include computer arithmetic, computer architecture, CAD, operating systems, and software engineering.