Modular Reduction by Multi-Level Table Lookup

Behrooz Parhami
Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106-9560, USA

Abstract

Common designs for reducing the lookup table size in modular reduction (computing of residues) all require peripheral logic in the form of multiplexers and/or (multi-operand) adders. We derive optimal two-level modular reduction circuits that are synthesized from lookup tables and pipeline latches only. We compare three such purely tabular realizations in terms of total table size. Extensions to more than two lookup levels, for gaining higher throughput, are also briefly discussed.

1. Introduction

Table lookup is an attractive method for function evaluation in VLSI signal processing applications since it leads to the replacement of irregular area-intensive random-logic structures with much denser memory arrays. Hence, increasingly, designs of computer arithmetic circuits are being based on table lookup, either as the primary computational scheme or for obtaining good initial approximations for speeding up iterative computations.

Modular reduction is the process of finding the remainder (residue) of a given input z with respect to a fixed modulus p. The computation of z mod p, or $|z|_p$, is needed in binary-to-residue conversion, as a final step within modular arithmetic operations, and for error checking based on residue codes. Many researchers have studied the use of tabular methods for modular reduction, especially in the context of binary-to-RNS (residue number system) conversion [1-4, 7-9, 13-14]. A commonly used method for this purpose is to obtain the residues corresponding to various segments of the input operand and then use a multi-operand modular adder to find the final residue [6, 10-11].

Let $m - 1$ be the largest possible value for the unsigned input z (i.e., $0 < z < m$). The binary representation of z thus has $b = \lceil \log_2 m \rceil$ bits. Residues mod p are represented as d-bit binary numbers ($d = \lceil \log_2 p \rceil$). We assume that tables of size $O(2^{2d}) = O(p^2)$ are practical. Our discussion thus does not cover the situation where fairly large (say 32-bit) residues are used to speed up arithmetic on huge (hundreds of bits long) numbers of the type needed for RSA encryption [5, 12].

The direct (single-level) table-lookup computation of z mod p, requiring a table of size

$$T_1 = dm \qquad (1)$$

is practical only when the range of z is fairly limited. Thus many approaches (e.g., based on truncated table lookup, piecewise table lookup, and stepwise refinement) have been suggested in order to reduce the required table size. As commonly implemented, these approaches require peripheral logic, in the form of multiplexers and adders, besides the lookup tables (see, e.g., [9]).

In this paper, we look at multi-level table lookup, without any additional logic, except for pipeline latches, as a way of reducing the total table size with respect to T_1. We derive and compare optimal two-level tabular schemes for computing z mod p which offer advantages for dense VLSI realizations. With the continual rise in memory density and throughput (e.g., by means of internal pipelining), such purely tabular realizations should become even more attractive in future.

2. Two-Stage Lookup with Truncation

Eqs. (2) and (3) below justify the truncated table-lookup approach that is the basis of some two-stage circuit implementations: A lookup table for the high-order $b - d + 1$ bits and a modular adder (built of an adder, a trial subtractor, and a 2-input mux) to incorporate the chopped low-order $d - 1$ bits.

$$z = 2^{d-1}\lfloor z/2^{d-1} \rfloor + z \bmod 2^{d-1} = 2^{d-1} z_{b-1:d-1} + z_{d-2:0} \qquad (2)$$

$$z \bmod p = (2^{d-1} z_{b-1:d-1} \bmod p + z_{d-2:0}) \bmod p \qquad (3)$$

The reason for the truncated segment being limited to $d - 1$ bits in this implementation is to ensure that the sum of this segment and the residue read out from the table is strictly less than $2p$, thus making it possible to compute the final residue by at most one trial subtraction and a selection.

In the pure table-lookup variant of this scheme, the above restriction is unnecessary. This leads to other implementations with a smaller initial table and a larger second-level table, thus offering additional opportunities for table size optimization.

0-7803-3694-1/97/$10.00 © 1997 IEEE

For example, if the truncated segment is d bits, then the first table is roughly halved in size while the size of the second table is doubled. Thus, this variant is always advantageous if the first table is larger than the second-level one. Since, we are free to choose any length for the truncated segment, we can easily choose a length that minimizes the total table size. The total table size when c bits are chopped (Fig. 1) is given by:

$$T_2 = d(\lceil m/2^c \rceil + 2^{c+d}) \quad (4)$$

Fig. 4 shows the total table size in this scheme for selected values of $m = 2^b$ and d. It is clearly seen that an optimal value for the number c of truncated bits exists in most cases.

Theorem 1: The optimal number c of truncated bits that minimizes the total table size given by Eq. (4) is within 1 of $(b - d)/2$, which is the exact optimal value when $b - d$ is even and $m = 2^b$.

The optimal value for the total table size T_2 in the special case of $m = 2^b$ is thus given by:

$$T_2^{\min} = d\, 2^{1+(b+d)/2} \quad (5)$$

3. Two-Stage Lookup with Segmentation

The b-bit input z can be broken into s segments of lengths $x_{s-1}, x_{s-2}, \ldots, x_1, x_0$, with $\sum_{i=0}^{s-1} x_i = b$, and the residue mod p of each segment found by consulting a table. The ith segment requires a table of height $v_i = 2^{x_i}$, the only exceptions being segment $s - 1$ which requires a table of height $\lceil m/2^{b-x_{s-1}} \rceil$ and segment 0 which does not require a table at all if $x_0 \le d$.

The s d-bit values read out from the tables must be added by using a modular multi-operand adder to obtain the final result z mod p. Thus, assuming non-tabular evaluation of the multi-operand modular addition, the total table size becomes

$$T_3^{\text{non-tab}} = d(\lceil m/2^{b-x_{s-1}} \rceil + \textstyle\sum_{i=1}^{s-2} 2^{x_i} + 2^{x_0}) \quad (6)$$

where the last term 2^{x_0} is included only if $x_0 > d$. In all published accounts of this method, the x_i are taken to be equal. While this assumption simplifies the description of the method and its analysis, it may not lead to the best hardware realization.

Theorem 2: In an optimal configuration, the segment lengths x_i and x_j cannot differ by more than 1 bit.

Thus, segment lengths should be made as equal as possible. Note that Eq. (6) is minimized if the x_i are all equal to 1. So it does not make sense to talk about optimizing the table size without also considering its effect on the complexity of the required multi-operand modular addition.

The pure two-level tabular version of this method performs the multi-operand modular addition via table lookup. Using a single table for the multi-operand modular addition (Fig. 2), we get:

$$T_3 = d(\lceil m/2^{b-x_{s-1}} \rceil + \textstyle\sum_{i=1}^{s-2} 2^{x_i} + 2^{x_0} + 2^{ds}) \quad (7)$$

Eq. (7) is much harder to analyze in general in view of its dependence on the special boundary conditions and varying segment lengths. However, taking advantage of Theorem 2, an approximate analysis assuming equal-size segments leads to the following result.

Theorem 3: The optimal number of segments in the above two-level arrangement is no greater than (but, for large b, asymptotically very close to) $\sqrt{b/d}$.

According to Theorem 3, the optimal configuration has approximately $\sqrt{b/d}$ segments of length $\sqrt{bd}$ bits, leading to:

$$T_3^{\min} \approx 2^{\sqrt{bd}}(\sqrt{bd} + d) \quad (8)$$

Fig. 5 shows the total table size in this scheme for selected values of $m = 2^b$ and d.

4. Two-Stage Stepwise Refinement

The stepwise refinement approach to modular reduction was first discussed in [9]. In this section, we focus on 2-stage refinement. In Stage 1, several high-order bits of z ($0 \le z < m$) are used to determine what negative multiple of p should be added to z to yield a result z' in the range $0 \le z' < m'$ with $p < m' < m$ and z mod $p = z'$ mod p. In Stage 2, the simpler computation z' mod p is performed.

Assuming $d' = \lceil \log_2 m' \rceil$, we obtain the following two-table scheme. The most significant $b - h$ bits of z, viz $z_{b-1:h}$, are used to access a w-word table ($w = \lceil m/2^h \rceil$) to obtain a d'-bit value. This value is the least significant d' bits of a negative multiple of p such that when added to z, the result z' satisfies $0 \le z' < m'$. After a d'-bit addition, whose carry out is ignored, the d'-bit sum is fed to an m'-word table to obtain the d-bit final result z' mod p.

The pure tabular version of the 2-stage refinement method replaces the adder and the second table above with a table of size $d\, 2^{2d'}$, leading to:

$$T_4 = d'(\lceil m/2^h \rceil + d\, 2^{2d'}) \quad (9)$$

Theorem 4: The total table size T_4 for the 2-stage pure tabular stepwise refinement method is always greater than the optimal table size $T_2^{\min}$ for the pure tabular truncation method.

Based on Theorem 4, we do not need to consider the 2-stage stepwise refinement method any further.

5. Comparison and Extensions

Fig. 6 shows a comparison of the two-level tabular truncation and segmentation methods for several example configurations. In fact, it is fairly easy to prove that in most practical situations, the two-level segmentation method has a lower total table size than the truncation method.

Having established the segmentation method as the best 2-level scheme, we next look for optimal designs with more than two levels of lookup tables. Generalizing the segmentation method to more than two levels is straightforward.

Consider the special case where the uniform segment size x at level 1 divides b. Then, at the output of level 1, we have bd/x bits which form the inputs to the second-level tables. Each subsequent level of tables then uses a segment size y to reduce the number of bits by a factor y/d. Since our goal is to reduce the number of bits to d eventually, the number of levels in the latter tree-structured reduction phase is

$$l \approx \log_2(b/x) \,/\, \log_2(y/d) \tag{10}$$

which leads to the total table size:

$$\begin{aligned} T_3 &\approx d((b/x)2^x + (2^l - 1)2^y) \\ &= d(2^x b/x - 2^y + 2^y (b/x)^{1/\log_2(y/d)}) \end{aligned} \tag{11}$$

The optimal values for x and y can be obtained by differentiating T_3 with respect to x and y, equating the derivatives with 0, and numerically solving the resulting system of two non-linear equations. Alternatively, T_3 can be written in terms of l and x (or l and y) as

$$\begin{aligned} T_3 &\approx d(2^x b/x + (2^l - 1)2^{d(b/x)^{1/l}}) \\ &= d((y/d)^l 2^{b(y/d)^{-l}} + (2^l - 1)2^y) \end{aligned} \tag{12}$$

with optimal values for l and x (y) sought as above. This multi-level table-lookup scheme can be easily pipelined with cycle time equal to that of a single table access. Thus, smaller tables lead to increased throughput but also imply higher latency due to an increase in the number $l + 1$ of levels.

The two special cases of $x = y = 2d$ and $x = y = d + 1$ are interesting: The former leads to $l + 1 = \log_2(b/d)$ levels and a total table size of $T_3 = (b - d)2^{2d}$; the latter yields the smallest component tables and thus the highest possible throughput. These two results can be used to bound an exhaustive search for an optimal solution using numerical methods.

The truncation and stepwise refinement methods can be similarly extended to more than two levels.

References

[1] Alia, G. and E. Martinelli, "A VLSI Algorithm for Direct and Reverse Conversion from Weighted Binary Number System to Residue Number System", *IEEE Trans. Circuit and Systems*, Vol. 31, pp. 1033-1039, 1984.

[2] Alia, G. and E. Martinelli, "VLSI Binary-Residue Converters for Pipelined Processing", *The Computer Journal*, Vol. 33, No. 5, pp. 473-474, 1990.

[3] Alia, G. and E. Martinelli, "On the Lower Bound to the VLSI Complexity of Number Conversion from Weighted to Residue Representation", *IEEE Trans. Computers*, Vol. 42, No. 8, pp. 962-967, Aug. 1993.

[4] Capocelli, R.M. and R. Giancarlo, "Efficient VLSI Networks for Converting an Integer from Binary System to Residue Number System and Vice Versa", *IEEE Trans. Circuits and Systems*, Vol. 35, pp. 1425-1430, Nov. 1988.

[5] Hung, C.Y. and B. Parhami, "Fast RNS Division Algorithms for Fixed Divisors with Application to RSA Encryption", *Information Processing Letters*, Vol. 51, pp. 163-169, 1994.

[6] Koc, C.K. and C.Y. Hung, "Multi-Operand Modulo Addition Using Carry-Save Adders", *Electronic Letters*, Vol. 27, No. 6, pp. 361-363, 15 Mar. 1990.

[7] Parhami, B., "Optimal Table-Lookup Schemes for Binary-to-Residue and Residue-to-Binary Conversions", *Proc. 27th Asilomar Conf. Signals, Systems, and Computers*, Pacific Grove, CA, Nov. 1993, Vol. 1, pp. 812-816.

[8] Parhami, B. and C.Y. Hung, "Optimal Table Lookup Schemes for VLSI Implementation of Input/Output Conversions and Other Residue Number Operations", *VLSI Signal Processing VII (Proc. of IEEE Workshop)*, La Jolla, CA, Oct. 1994, pp. 470-481.

[9] Parhami, B., "Analysis of Tabular Methods for Modular Reduction", *Proc. 28th Asilomar Conf. Signals, Systems, and Computers*, Pacific Grove, CA, Oct./Nov. 1994, pp. 526-530.

[10] Piestrak, S.J., "Design of Residue Generators and Multi-Operand Modular Adders Using Carry-Save Adders", *Proc. Int'l Symp. Computer Architecture*, May 1991, pp. 100-107.

[11] Piestrak, S.J., "Design of Residue Generators and Multioperand Modular Adders Using Carry-Save Adders", *IEEE Trans. Computers*, Vol. 43, No. 1, pp. 68-77, Jan. 1994.

[12] Posch, K.C. and R. Posch, "Modulo Reduction in Residue Number Systems", *IEEE Trans. Parallel and Distributed Systems*, Vol. 6, No. 5, pp. 449-454, May 1995.

[13] Shenoy, A.P. and R. Kumaresan, "Fast Base Extension Using a Redundant Modulus in RNS", *IEEE Trans. Computers*, Vol. 38, No. 2, pp. 292-297, Feb. 1989.

[14] Soderstrand, M.A., W.K. Jenkins, G.A. Jullien, and F.J. Taylor (Editors), *Residue Number System Arithmetic*, IEEE Press, New York, 1986. (See, in particular, Papers 2-6, 3-14, and 3-16).

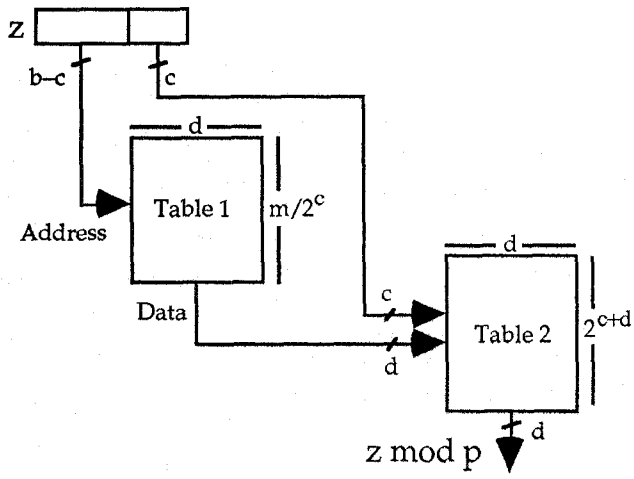

Fig. 1. Table-lookup with truncated operand.

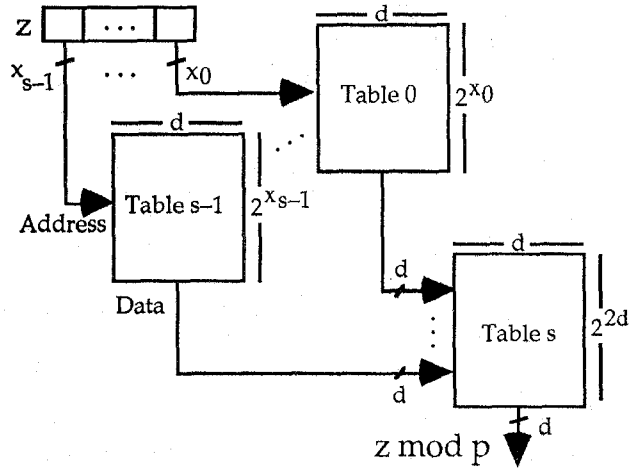

Fig. 2. Piecewise table-lookup based on segmentation.

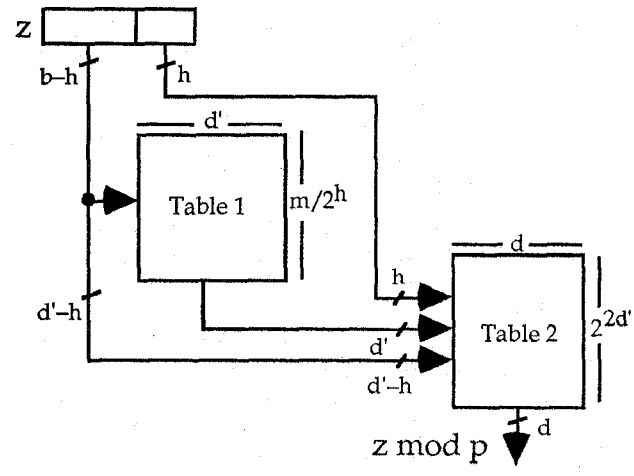

Fig. 3. Table-lookup based on stepwise refinement.

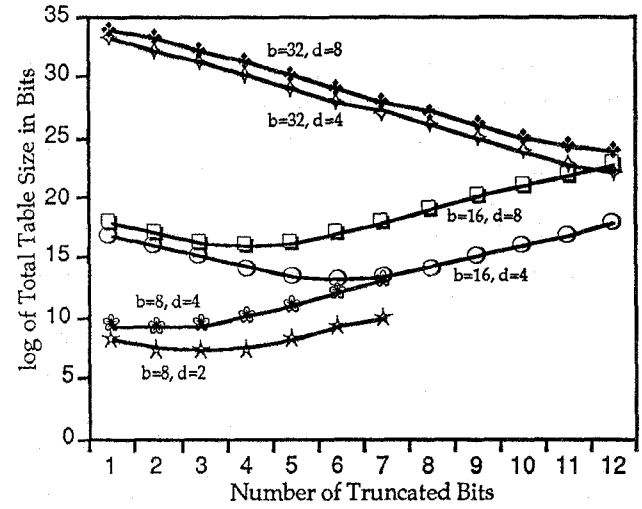

Fig. 4. Modular reduction with 2-level pure table-lookup based on truncation.

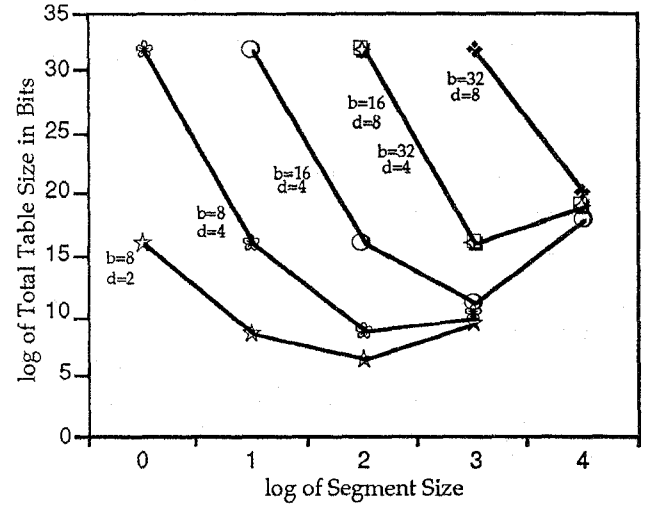

Fig. 5. Modular reduction with 2-level pure table-lookup based on segmentation.

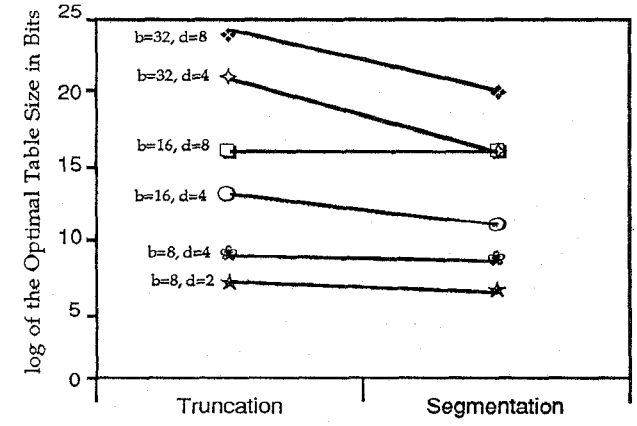

Fig. 6. Comparison of truncation and segmentation methods in their optimal versions.