Residue Arithmetic and VLSI

Chao-Lin Chiang
and
Lennart Johnsson

Computer Science Department
California Institute of Technology

5092:TR:83

RESIDUE ARITHMETIC AND VLSI

Chao-Lin Chiang and Lennart Johnsson

Computer Science
California Institute of Technology
Pasadena CA 91125

5092:TM:83

Presented at
1983 IEEE International Conference on
Computer Design: VLSI in Computers (ICCCD'83)
New York,
Oct. 31-Nov 3, 1983

The research described in this paper
was sponsored in part
by the Defense Advanced Research Projects Agency,
ARPA Order number 3771, and monitored by the
Office of Naval Research
under contract number N00014-79-C-0597

RESIDUE ARITHMETIC AND VLSI

Chao-Lin Chiang and Lennart Johnsson
Computer Science
California Institute of Technology
Pasadena, CA. 91125

Abstract

In the residue number system arithmetic is carried out on each digit individually. There is no carry chain. This locality is of particular interest in VLSI. An evaluation of different implementations of residue arithmetic is carried out, and the effects of reduced feature sizes estimated.

At the current state of technology the traditional table lookup method is preferable for a range that requires a maximum modulus that is represented by up to 4 bits, while an array of adders offers the best performance for 7 or more bits. A combination of adders and tables covers 5 and 6 bits the best. At 0.5 μm feature size table lookup is competitive only up to 3 bits. These conclusions are based on sample designs in nMOS.

1. INTRODUCTION

Weighted number systems such as the binary number system and the decimal number system have a carry chain. It is often limiting the performance of computer arithmetic. In the residue number system [1], a number is represented by several residue digits. The arithmetic operations addition and multiplication are performed on each digit independently. This local property allows a high degree of concurrency. However, operations such as division, magnitude comparison and overflow detection do not have the local property, [4], [6], [10].

The residue digits are obtained by evaluating a number X modulo a set of integers $m_1, m_2, ..m_p$, which are pairwise relatively prime, i.e., $\gcd(m_i, m_j) = 1$, for $i \neq j, 1 \leq i, j \leq p$. According to the Chinese Remainder Theorem [3], there exist a unique residue representation $(x_1, .., x_p)$ for any number X in the range $[0, R-1]$,

$$\text{where} \quad R = \prod_{i=1}^{p} m_i \quad \text{and} \quad x_i = X \bmod m_i$$

A number represented in residue code can be converted to binary code by the formula:

$$X = \left| \sum_{i=1}^{p} x_j \cdot \hat{m}_j \cdot \left| \frac{1}{\hat{m}_j} \right| \right| \mod R$$

where:

$$\hat{m}_j = \frac{R}{m_j} \quad \text{and} \quad \left| \frac{1}{\hat{m}_j} \right| = a \text{ iff } (\hat{m}_j \cdot a) \bmod R = 1$$

Let X and Y have residue codes $(x_1, .., x_p)$ and $(y_1, .., y_p)$, and be such that $X, Y, X+Y, X \cdot Y \in [0, R-1]$. Then

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

and it follows that

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

The smaller the range of each digit is, the higher is the degree of concurrency. Minimizing the maximum modulus for a given range R maximizes the concurrency. In Table 1, M is the number of bits required for a binary encoding of a digit in the residue representation of a number. N is the number of bits required for a direct binary encoding of the same number. All moduli are of the form p^k, where p is prime. Moduli are listed in order of increasing p.

M	moduli	N	$\frac{\log N}{M}$
3	4,3,5,7	8.71	1.04
4	15,9,5,7,11,13	18.46	1.05
5	16,27,25,7,11,13,17,19,23,29,31	46.04	1.10
6	64,27,25,49,11,13, ... ,53,59,61	~90	1.08
8	256,243,125,49, ... ,239,241,251	~368	1.06

Table 1. Maximum number of bits in residue code (M) and bits in binary code (N)

It can be proved [14] that $(\log N)/M$ is asymptotically approaching 1, i.e., $M = O(\log N)$. The significance of this result is that instead of using N-bit arithmetic, $\log N$-bit binary arithmetic suffice. (Each number x_i, y_i is represented in binary code.) However, $O(N/\log N)$ such units are required instead of one N-bit unit. Also, addition and multiplication is more complex than in the binary number system in that it is performed modulo the set of integers, $m_1, .., m_p$.

2. IMPLEMENTATIONS OF RESIDUE ARITHMETIC

Addition and multiplication modulo the integers m_1, $m_2, .., m_p$ are traditionally implemented by table lookup [7], [11], [12], [13]. The table size grows exponentially with the number of bits required to represent the max-

imum modulus. Storage can be made very dense, but a large storage contains long wires unless it is hierarchical [9]. The wire delay does not decrease with feature size, and eventually becomes a performance limiting factor. A few alternatives to the table lookup method are discussed below.

2.1 Residue Adder

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

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

One M-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 selects the correct output, Figure 1.

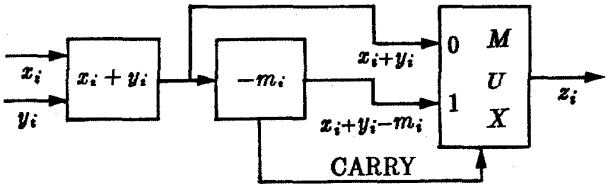

Figure 1. An implementation of residue addition

2.2 Residue Multiplier

The following three different methods to implement residue multipliers have been evaluated:

1. pure table lookup
2. combinations of tables and adders
3. arrays of adders

2.2.1 Pure Table Lookup

A table for a residue multiplier has two M-bit inputs, x_i and y_i, and produces one M-bit output, $z_i = (x_i \cdot y_i) \bmod m_i$, Figure 2. The total number of bits in the table is $2^{2M}M = 4^M M$, $M \geq \log m_i$.

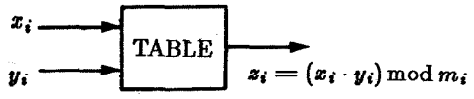

Figure 2. Table lookup residue multiplier

2.2.2.a Quarter Square Residue Multiplier [2]

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" $|\frac{(\cdot)^2}{4}|_{m_i}$ is implemented by lookup table. Each table has $2^{M+1}M = 2M2^M$ bits. One M-bit binary adder/subtractor and one M-bit residue subtractor are needed.

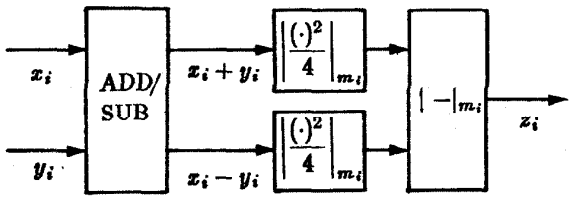

Figure 3. Quarter square residue multiplier

2.2.2.b Index Transform Residue Multiplier

One familiar method of multiplication is "log transform – addition – antilog transform"[5]. Similarly, residue multiplication can be performed by "index transform – residue addition – reverse index transform". The index transform IND_r is defined by [3]:

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

An implementation based on the equation

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

is shown in Figure 4. The index and reverse index transform is implemented by tables. There are two tables for the index transform and one for the reverse index transform. Each table has $M2^M$ bits.

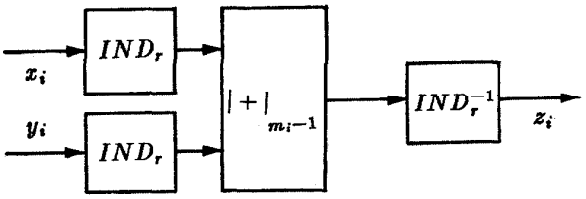

Figure 4. Index transform residue multiplier

2.2.3 Array Multiplier

A residue array multiplier can be 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 $O(M)$ and an area of $O(M^2)$. The complexity of the conversion part can be made equal to that of the multiplier by employing carry save adders [8].

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

$$a_{2M-1}2^{2M-1} + \ldots + a_M 2^M + \ldots \bmod m_i =$$
$$= a_{2M-1}(2^{2M-1} \bmod m_i) + \ldots + a_{2M}(2^{2M} \bmod m_i) + \ldots$$

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

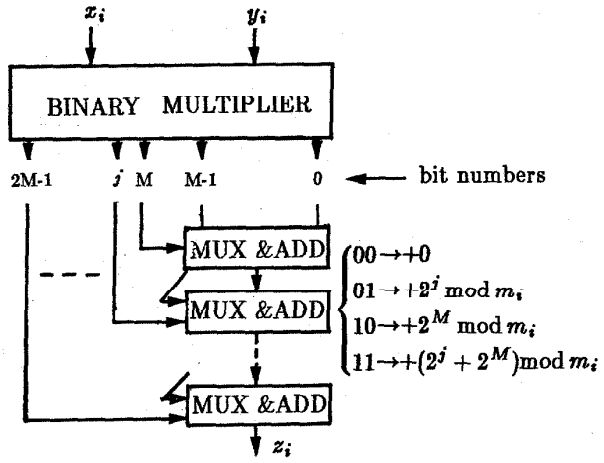

Figure 5. Residue array multiplier

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

2.3 Analysis of different multiplier schemes

Some features of the four residue multiplier designs discussed above are summarized in Table 2. From the 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 others depending on the value of M. 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	Bin. adder	Res. adder
2.2.1	$M4^M$	0	0
2.2.2.a	$4M2^M$	M	M
2.2.2.b	$3M2^M$	0	M
2.2.3	0	$2M^2$	0

Table 2. Some features of residue multipliers

3. RESIDUE MULTIPLIER IN nMOS

3.1 Delay and area estimates

Tables 3 and 4 contain delay estimates for residue multiplier designs in $4\mu m$ and $0.5\mu m$ nMOS. The delay estimates are based on a τ-model [9], further developed to cover PLA's [15]. From the tables it is concluded that implementations based on table lookup do not scale well. The delay decreases only slightly. 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 as the feature sizes are reduced. The areas, Table 5, expressed in terms of λ^2 [9] is to first order independent of the feature size.

Bit	Table	Ind. tran.	Array
3	198	370	900
4	400	480	1200
5	1070	700	1500
6	2400	1270	1800
7	6600	2000	2100
8	26000	5600	2400

($unit: \tau$, $1\tau \approx 0.2ns$)

Table 3. Delay in residue multipliers for $4\mu m$ nMOS

Bit	Table	Ind. tran.	Array
3	236	372	900
4	640	485	1200
5	2040	710	1500
6	7600	1400	1800
7	31000	2200	2100
8	150000	6100	2400

($unit: \tau$, $1\tau \approx 0.025ns$)

Table 4. Delay in residue multipliers for $0.5\mu m$ nMOS

From the tables we conclude that at $4\mu m$ feature size table lookup is preferable both with respect to area and delay for $M \leq 4$ bits. The delay of the index transform method is less than the other two methods for $5 \leq M \leq 7$ bits, while the array multiplier needs an area equal to or less than the other two for $M \geq 5$ bits. At $0.5\mu m$ feature size the table lookup method looses its competitive edge at $M = 3$.

Bit	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

($unit: 1000\lambda^2$)

Table 5. Chip area of residue multipliers

3.2 Chip characteristics

Figure 6 shows the design of a residue array multiplier. A 4-bit version of the multiplier requires an area of $300 \times 700\lambda^2$, and an 8-bit version $540 \times 1200\lambda^2$. The designs are submitted for fabrication through the DARPA MOSIS foundary service.

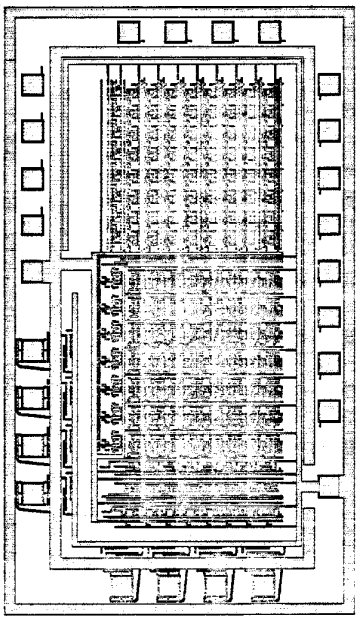

Figure 6. The metal layer of an 8-bit residue multiplier

Simulations indicates that carry and sum bit generation require 10∼15 ns. The critical path in each M-bit residue multiplier goes through $3M$ adders. Therefore, the total delay is $30M$ to $45M$ ns, and hence about 240∼360 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.

4. CONCLUSION

The evaluation of residue multiplier designs in nMOS shows that at the current state of the technology, implementations based on table lookup are preferable both with respect to performance and area for a range that can be encoded with 18 bits in a binary number system. If a larger range is needed, index transform and array designs both 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 for a binary encoding. However, if the 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 lookup designs are preferable only for ranges of up to 9 bits, when binary encoded.

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.

Acknowledgment

This work was supported by the Defense Advanced Research Project Agency (DARPA) under contract N00014-79-C-0597 with the California Institute of Technology. Views and conclusions contained in this paper are the authors and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government or any agency connected with them.

References

[1] H. Garner, *"The Residue Number System"*, IRE Tran. on Electronic Computers, June 1959

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

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

[4] D. Banerji, J. A. Brzozowski, *"Sign Detection in Residue Number Systems"*, IEEE C-18, Apr. 1969

[5] N.G. Kingsburg and P.J. Rayner, *"Digital Filtering Using Logarithmic Arithmetic"*, Electron. Lett., vol.7, Jan., 1971

[6] E. Kinoshita, H. Kosako and Y Kojima, *"General Division in Symmetric Residue Number Systems"*, IEEE C-22 Feb. 1973

[7] W. Jenkins, *"The Use of Residue Number Systems in the Design of Finite Impulse Response Digital Filters"* IEEE, CAS-24 Apr. 1977.

[8] K. Hwang, *"Computer Arithmetic"*, Wiley 1979

[9] Mead & Conway, *"Introduction to VLSI System"*, Addison-Wesley, 1980

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

[11] C. H. Huang, D. Peterson, *"Implementation of a Fast Digital Processor Using the Residue Number System"*, IEEE, CAS-28 Jan. 1981

[12] 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

[13] 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

[14] F. Barsi, A. Di Cola, *"A VLSI Binary Multiplier Using Residue Number Systems"*, Proceeding ICCC, Oct 1982

[15] C. Chiang, *"Distributed RC delay line model and MOS PLA timing estimation"*, Technical Report, California Institute of Technology, in preparation.