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

© Copyright 1983 California Institute of Technology
RESIDUE ARITHMETIC AND VLSI

Chao-Lin Chi 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 \( \mu m \) 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], [8], [10].

The residue digits are obtained by evaluating a number \( X \) modulo a set of integers \( m_1, m_2, \ldots, m_p \), 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 \( (x_1, \ldots, x_p) \) for any number \( X \) in the range \( [0, R - 1] \),

where \( R = \prod_{i=1}^{p} m_i \) and \( x_i = X \mod m_i \).

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

\[
X = \prod_{i=1}^{p} x_i \cdot \frac{1}{\hat{m}_j} \mod R
\]

where:

\[
\hat{m}_j = \frac{R}{m_j} \text{ and } \frac{1}{\hat{m}_j} = a \text{ iff } (\hat{m}_j \cdot a) \mod R = 1
\]

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

\[
|X + Y|_{m_i} = |x_i + y_i|_{m_i} \text{ and } |X \cdot Y|_{m_i} = |x_i \cdot y_i|_{m_i}
\]

and it follows that

\[
(|X + Y|_{m_1}, \ldots, |X + Y|_{m_p}) = (|x_1 + y_1|_{m_1}, \ldots, |x_p + y_p|_{m_p})
\]

\[
(|X \cdot Y|_{m_1}, \ldots, |X \cdot Y|_{m_p}) = (|x_1 \cdot y_1|_{m_1}, \ldots, |x_p \cdot y_p|_{m_p})
\]

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 \( p \).

<table>
<thead>
<tr>
<th>( M )</th>
<th>moduli</th>
<th>( N )</th>
<th>( \log N )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4, 3, 5, 7</td>
<td>8.71</td>
<td>1.04</td>
</tr>
<tr>
<td>4</td>
<td>15, 9, 5, 7, 11, 13</td>
<td>18.46</td>
<td>1.05</td>
</tr>
<tr>
<td>5</td>
<td>16, 27, 25, 7, 11, 13, 17, 19, 23, 29, 31</td>
<td>46.04</td>
<td>1.10</td>
</tr>
<tr>
<td>6</td>
<td>34, 49, 55, 40, 11, 13, 39, 25, 62, 61</td>
<td>53, 50, 51, 38</td>
<td>90</td>
</tr>
<tr>
<td>8</td>
<td>256, 243, 125, 48, 49, 339, 241, 251</td>
<td>~368</td>
<td>1.06</td>
</tr>
</tbody>
</table>

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 \( x_i \) 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, \( m_1, \ldots, m_p \).

2. IMPLEMENTATIONS OF RESIDUE ARITHMETIC

Addition and multiplication modulo the integers \( m_1, m_2, \ldots, m_p \) 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-
2.1 Residue Adder

Residue addition $z_i = (x_i + y_i) \mod m_i$, 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 $z_i + y_i$ while another computes $z_i + y_i - m_i$. 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](image1)

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, $z_i = (x_i \cdot y_i) \mod m_i$, Figure 2. The total number of bits in the table is $2^M \cdot M = 4^M M$. $M \geq \log m_i$.

![Figure 2. Table lookup residue multiplier](image2)

2.2.2a Quarter Square Residue Multiplier [2]

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

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

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

![Figure 3. Quarter square residue multiplier](image3)

2.2.2b Index Transform Residue Multiplier

One familiar method of multiplication is "log transform – addition – antilog transform"[6]. Similarly, residue multiplication can be performed by "index transform – residue addition – reverse index transform". The index transform $IND_r$ is defined by [3]:

$$IND_r C \equiv b \mod P \iff \phi_r \equiv C \mod P, \ P \ \text{prime}$$

An implementation based on the equation

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

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](image4)

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 [9].

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

$$a_{2M-1} 2^{2M-1} + ... + a_0 2^M + ... \mod m_i =
= a_{2M-1}(2^{2M-1} \mod m_i) + ... + a_0(2^M \mod m_i) + ...
$$

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 $(2^{2M-1} \mod m_i), (2^{2M-2} \mod m_i), ..., (2^M \mod m_i)$ shall be added to the $M$ lowest order bits, Figure 5.
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 2. Some features of residue multipliers |
|-----------------|-----------------|-----------------|
|                | Table | Bin. adder | Res. adder |
| 2.2.1          | $M^4$ | 0           | 0           |
| 2.2.2.a        | $4M^2$ | $M$        | $M$        |
| 2.2.2.b        | $3M^2$ | 0           | $M$        |
| 2.2.3          | $2M^2$ | 0           | 0           |

3. RESIDUE MULTIPLIER IN nMOS

3.1 Delay and area estimates

Tables 3 and 4 contain delay estimates for residue multiplier designs in 4μm and 0.5μm 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 $\tau$ it increases significantly. The index transform approach employs much smaller tables. The delay measured in $\tau$ 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 area, Table 5, expressed in terms of $λ^2$ [9] is to first order independent of the feature size.

From the tables we conclude that at 4μm 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 0.5μm feature size the table lookup method looses its competitive edge at $M = 3$.

| Table 4. Delay in residue multipliers for 0.5μm nMOS |
|-----------------|-----------------|-----------------|
| Bit | Table | Ind. tran. | Array |
| 3   | 189   | 370         | 900   |
| 4   | 400   | 480         | 1000  |
| 5   | 1070  | 700         | 1500  |
| 6   | 2400  | 1270        | 1800  |
| 7   | 6600  | 2000        | 2100  |
| 8   | 28000 | 5600        | 2400  |

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

| Table 5. Chip area of residue multipliers |
|-----------------|-----------------|-----------------|
| Bit | Table | Ind. tran. | Array |
| 3   | 45    | 95          | 115   |
| 4   | 147   | 188         | 207   |
| 5   | 524   | 282         | 297   |
| 6   | 1070  | 543         | 407   |
| 7   | 8000  | 1007        | 591   |
| 8   | 32000 | 2177        | 664   |

(unit : 1000$λ^2$)

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 500 $\times$ 700$λ^2$, and an 8-bit version 540 $\times$ 1200$λ^2$. The designs are submitted for fabrication through the DARPA MOSIS foundry service.
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