# BINARY PARADIGM AND SYSTOLIC ARRAY IMPLEMENTATION FOR RESIDUE ARITHMETIC.

David Y. Y. Yun

Chang N. Zhang

Department of Computer Science and Engineering

Southern Methodist University

Dallas, Texas 75275

**abstract**

The problem of residue, or modular, arithmetic is fundamental to symbolic and algebraic computation, coding theory and applications, as well as to error-free arithmetic computations. This paper describes novel algorithms that can lead to efficient hardware for arithmetic operations in residue domains. One of the main achievements is in allowing the flexibility of changing moduli. The technology of systolic array has been used to implement one of the most representative operations, the modular multipler. It is shown that a linear systolic array can compute N modular products in time O(N) with constant number of cells.

## 1. introduction

Residue, or modular, arithmetic is well known to be of fundamental to symbolic and algebraic computation, coding theory and applications, as well as to error-free arithmetic computations. It has long been recognized that residue arithmetic is particularly suitable for parallel computations, by either software or hardware implementations. Most implementations of residue arithmetic algorithms are by software only, and they do not take full advantage of machine instructions or hardware. Those algorithms for residue arithmetic, which attempt to improve computational speed by hardware, often involve trade-off with other difficulties. They are sometimes memory intensive in that some table lookup operations are used to perform the arithmetic [2,3]. Other times, these implementations primarily rely on using moduli of the form $2^n, 2^n-1$ and $2^n+1$ [1,2]. These algorithms essentially take advantage of the inherent binary number representation and hardware instructions to offer computational efficiency. Unfortunately, these restricted powers-of-two moduli are frequently inadequate when wider selections of moduli are required for computation in several fields. Residue arithmetic is one such example field, which clearly need more moduli than these restricted powers-of-two. Often, the further requirement that these moduli be primes makes it nearly impossible to use these restricted powers-of-two. Our work eliminates these restrictions for the moduli, while still take advantage of the inherent binary representations and instructions. It increases modular computational efficiency either by directly using machine instructions or by special hardware designs.

In multiple-moduli arithmetic, let $\{p_1, \ldots, p_l\}$ be a set of relative prime integers (moduli) and $x \in Z_P (0 \leq x < P)$, where $P = \prod_{i=1}^{l} p_i$ then there exist integers $k_i$ and $x_i$ such that

$$x = k_i p_i + x_i, \quad i = 1,2,\ldots,l$$

The i-th residue of $x$, $x \bmod p_i$, is denoted by $x_i$. $x$ can be uniquely represented by a $l$-tuple: $x = (x_1, x_2, \cdots, x_l)$. We use $=_P$ to denote modular equivalence.

If $x, y \in Z_P$ and $x = (x_1, x_2, \cdots, x_l)$, $y = (y_1, y_2, \cdots, y_l)$,

then $Z = x \; o \; y = (z_1, z_2, \cdots, z_l)$, where $z_i =_{p_i} x_i \; o \; y_i$ for $i = 1,2, \cdots l$, and $o$ denotes the operation *, + or -. It is clear that the suboperation in each modulus is independent of the other. Thus, parallel algorithm and architecture can be designed to process all modular operations concurrently.

In this paper we propose algorithms that use only binary addition and multiplication operations for residue arithmetic with arbitrarily selectable moduli. For residue addition, the algorithm needs one binary addition cycle time ( as opposed to 2 binary addition cycle times in the normal method). For residue multiplication, the algorithm needs a constant number of normal binary multiplications and additions for moduli of form $2^n + s$ and $2^n - s$, where s is chosen from



CH2146-9/85/0000/0189$01.00 

some scale, such as $s \leq 2^{n/2}$. In that case, $xy$ mod $2^n - s$ can be computed in four multiplications. We also present a linear systolic array that computes a sequence of N residue products, which requires O(N) time units with constant number of basic cells. The unit time is that of one binary word multiplication.

## 2. modular addition

First, we consider moduli of the form $p=2^n-s$.
The problem of the modular addition is that:
The normal way to compute $x + y$ mod $p$ is first to compute $c := x + y$. One then computes $c - p$. If there is no underflow, this is the answer, also $c$ is the answer. So it needs two binary addition cycle times.
The following algorithm only needs one binary addition cycle time ($p = 2^n - s$).

$c := x+y+s$; $d := x + y$;

*if* bit $n$ of $c = 1$ *then* *return* $(c)_{0-n-1}$ ( truncated to bits 0... $n$-1 ) ;

*else* *return* $d$ ;

Note that the additions on the first line can be done in parallel and the addition that has three addends can be implemented by a multioperand adder [8].
It is easy to show that the algorithm actually computes $x+y$ mod $p$.
In fact, note $s \leq x+y+s < 2^{n+1} - s$, therefore if $x+y+s \geq 2^n$ then
$x + y = _p (x + y + s) - s - p = (x + y + s) - 2^n < p$
otherwise if $(x + y + s) < 2^n$
then $x + y = _p (x + y + s) - s < p$.

Next, we consider moduli of form $p = 2^n + s$. Our algorithm quite clearly becomes:

$c := x + y - s$ ; $d := x + y$;

*if* $n$th bit of $c = 1$ or $n$+1th bit of $c = 1$ *then* *return* $c - 2^n$

*else* *return* $d$ ;

Note that $-s \leq c < 2^{n+1} + s$.

Therefore, if $c \geq 2^n$ ($c_n = 1$ *or* $c_{n+1} = 1$)

then $x + y = _p (x + y - s) - p + s$

$= ( x + y - s ) - 2^n < p$

otherwise if $c < 2^n$

## 3. modular multiplication

### 3.1 modulus $p = 2^n - s$

Definition 1: Let $x$ be an integer and $p$ be a moduli of form $p = 2^n - s$. Define a function $F$ such that $F(x) = x_0 + x_1 s$ where $x = x_0 + x_1 2^n$.

Theorem 1: $F(x) = _p x$.

Proof: $F(x) = x_0 + x_1 s$

$= x_0 + x_1(2^n - p)$

$= _p x_0 + x_1 2^n = x$ $\Delta$

Example: Let $n=3$ and $s=3$ then $p = 2^3 - 3 = 5$.

$x = 19 = 2 \times 2^3 + 3 = _p 4$

$x_0 + x_1 s = 3 + 2 \times 3 = _p 4$

Theorem 2: Let $x, y \in Z_p$ and $p$ be modulus of form

$p = 2^n - s$, where $s = 2^{(1-2^{-k})n}$, $k$ is a positive constant integer.

then $F^{2^k}(xy) < 2^{n+1}$,

where $F^{2^k}(xy) = \underbrace{F(F(\cdots(F}_{2^k}(xy))\cdots))$.

Proof: Let $w = xy$, $x, y \in Z_p$ we have $0 \leq w < 2^{2n}$

Let $w = w_0^0 + w_1^0 2^n$ *where* $0 \leq w_0^0 < 2^n$, $0 \leq w_1^0 < 2^n$ .

From definition 1, $F(w) = w_0^0 + w_1^0 s$ where

$0 \leq F(w) < 2^n + 2^{(2-2^{-k})n}$.

$F(w)$ can be rewritten as $F(w) = w_0^1 + w_1^1 2^n$ **where**

$0 \leq w_0^1 < 2^n$ $0 \leq w_1^1 < 2^{(1-2^{-k})n}$.

**By the inductive hypothesis:**
assuming that $F^i(w) < 2^n + 2^{(2-2^{-k}\,i)n}$. we have

$F^i(w) = w_0^i + w_1^i 2^n$

where $0 \leq w_0^i < 2^n$, $0 \leq w_1^i < 2^{(1-2^{-k}\,i)n}$.

Consider $F^{i+1}(w) = w_0^i + w_1^i s$, then

$0 \leq F^{i+1}(w) < 2^n + 2^{(2-2^{-k}(i+1))n}$.

Therefore, we have

$F^{2^k}(xy) < 2^n + 2^n = 2^{n+1} (i = 2^k)$ $\Delta$

Note that $xy \equiv_p F^{2^k}(xy) < 2^{n+1}$ (by theorem 1 & 2)

and $F^{2^k+1} < 2^n + s$.

Example: Let $n=4$ and $s=4$ ( $k=1$) then $p = 2^4 - 4 = 12$.

$x = 10$, $y = 11$, $xy = 110 \equiv_p 2$.

$F(xy) = 6 \times 4 + 14 = 38$ $(110 = 16 \times 6 + 14)$

$F^2(xy) = 2 \times 4 + 6 = 14$ $(38 = 16 \times 2 + 6)$

$F^3(xy) = 14 \equiv_p 2$

We can now use the following algorithm for residue multiplication(asumming $s \leq 2^n/3$).

$c := x\ y$;

*for* $i := 1$ to $2^{k+1}$ *do* $c := F(c)$ ;

$d := F(c + s)$ ;

*if* $d \geq 2^n$ *then return* $d - 2^n$

*else return* $c$ ;

Note that $0 \leq F^{2^{k+1}}(xy) < 2^n + s$ and let

$d = F^{2^{k+1}}(xy) + s$. We have that $s \leq d \leq 2^n + 2s$

Because $xy \equiv_p d - s$, if $d \geq 2^n$ then

$xy \equiv_p d - s - p = d - 2^n \leq p$ $(s \leq 2^n/3)$.

Therefore,

$$xy \equiv_p \begin{cases} d - 2^n & \text{if } d \geq 2^n \\ c & \text{if } c < 2^n \end{cases}$$

**3.2 modulus $p = 2^n + s$**

Numbers in this $Z_p$ can be represented as binary

$(n+1)$- tuples: $x = \sum_{i=0}^{n} x_i 2^i$.

Defining a function $G$ , we can get some facts which are similar to the ones of modulus $p = 2^n - s$ discussed above.

Definition 2: Define a function $G$ such that $G(x) = x_0 - x_1 s$, where $x_0 + x_1 2^n = x$

Theorem 3: $G(x) \equiv_p x$

Theorem 4: Let $x$ , $y \in Z_p$, $p = 2^n + s$, $s = 2^{(1-2^{-k})n}$, $k$ is a positive integer, then

$$-2^n < G^{2^k}(xy) < 0 \text{ or } 0 \leq G^{2^k}(xy) \leq 2^{n+1}.$$

The corresponding algorithm of multiplication is as follows(asumming $s \leq 2^n/3$):

$c := x\ y$ ;

*for* $i = 1$ to $2^{k+1}$ *do* $c := G(c)$ ;

*if* $c < 0$ *then return* $c + p$

*else if* $c > p$ *then return* $c - p$

*else return* $c$ ;

The proof of the theorems 3 and 4 is similar to the previous ones, and the correctness of the algorithm is also straightforward.

## 4. systolic array implementation

To ensure efficiency in the algorithms described above, the set of moduli should be carefully chosen. A particularly useful set is $\{2^n - 1, 2^n, 2^n + 1\}$, which is exactly the special case of $k = 0$ , $s = 1$. Another useful example is to choose a set of moduli of form $p_i = 2^n - s_i$ or $p_i = 2^n + s_i$, $i = 1,2, \cdots, l$, where $s_i \leq 2^{n/2}$ ( $k = 1$). By the Theorem 2 & 4, a multiplication of two integers in these moduli needs four binary multiplications.

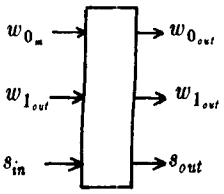
One of the hardware implementations of these algorithms is to use systolic array. We present a linear systolic array implementation to accomplish this process. The problem of computing N modular products can be defined as follows:

given $x_i$ , $y_i \in Z_p$ where $i = 1,2, \cdots, N$,

compute $Z_i = x_i\ y_i\ mod\ p$ $i = 1,2, \cdots, N$,

where $p = 2^n - s$ , $s \leq 2^n/3$ .

### 4.1 two basic cells

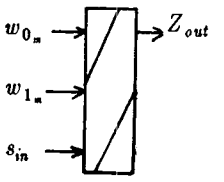



$s_{out} := s_{in}$;

$w_{0_{out}} := ( w_{1_{in}} s + w_{0_{in}} )_{0-(n-1)}$ (bits 0 ... $n-1$)

$w_{1_{out}} := ( w_{1_{in}} s + w_{0_{in}} )_{n-(2n-1)}$ (bits $n$ ... $2n-1$)

Figure 1 basic cell(a).




$$Z_{out} := \begin{cases} ( w_{0_{in}} + s )_{0-(n-1)} & \text{if } ( w_{1_{in}} )_n = 1 \\ & \text{or } ( w_{0_{in}} + s )_n = 1 \\ w_{0_{in}} & \text{otherwise.} \end{cases}$$

Figure 2 basic cell(b).

### 4.2 the array

A full array consists of five cells(the right-most cell is basic cell(b)) as depicted if Figure 3. The input $w_{0_{in}}$, $w_{1_{in}}$ and $s_{in}$ to the left-most cell is 0, $x_i$ and $y_i$ respectively, and the input, $s_{in}$, of the second left-most cell is $s$.

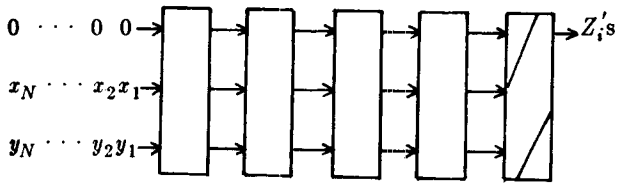



Figure 3.

### 4.3 an illustration

Consider the case N = 6. Define a cycle to be the time to perform the function of the basic cells. The status of the array at the end of the next five cycles are depicted in Figure 4.

### 4.4 remarks

A systolic architecture is designed to perform the multiplication operation in the moduli $p=2^n - s$. This multiplier requires 5 basic cells. This multiplier is highly suitable for VLSI implementation because of its simple and local comtrol methodology. Note that if we choose the form of moduli $p = 2^n - s$ where $s$ is the form of $2^k$ ,$2^k + 1$ or $2^k - 1$ , then the operations of the basic cells are only shift and additions(subtractions).

In general the basic cell(a) needs a $n*m$-bit multipler (assuming $m=2n$). Note that the output of the cell, $w_{0_{in}}$ and $w_{1_{in}}$, is two n-bit integers so we can divide its input, $w_{1_{in}}$, into two parts:

$$w_{1_{in}} = w_{1_{in\,H}}\, 2^m + w_{1_{in\,L}}$$

then compute $w_{1_{in\,H}} * s$ and $w_{1_{in\,L}} * s$ separately by two $m*m$-bit multiplers and group them into two $n$-bit sections as outputs:

$w_{0_{out}}$ ( low section 0-n-1 bits ) and

$w_{1_{out}}$ ( high section n-2n-1 bits ).

The detail design and more applications of these processes were reported in another paper [6].

### refences


[1] F. J. Taylor, "A VLSI Residue Arithmetic Multiplier", IEEE Trans. comput. Vol. c-31, pp. 540-546, June 1982.

[2] W. J. Jenkins, "A Highly Efficient Residue-Combinatorial Arithitecture Digital Filters", IEEE Vol. 66, pp. 700-702, June 1978.

[3] W. K. Jenkins and F. J. Leon, "The Use Of Residue Number System In The Design Of finite Impulse response Filters", IEEE Trans. Circuits Syst., Vol. CAS-24, Apr. 1977.

[4] J. L. Massey and O. N. Garcia, "Error Correcting Codes in Computer Arithmetic", Vol. 4, Plenum Press, New York, 1971.

[5] H. T. Kung, "Why Systolic Architectures", IEEE Computers, Vol. 15, No. 1, pp. 37-46, Jan. 1982.

[6] D. Y. Y. Yun and C. N. Zhang. "Systolic Array For Programmable Galois Field Multiplier", Report Dept. of Comp. Sc. Eng. Southern Methodist University, Oct. 1984.

[7] John D. Lipson, "Elements Of Algebra And Algebraic Computing", Addison-Wesley Publishing Company, 1981.

[8] Kai Hwang, "Computer Arithmetic: Principles, Architecture And Design", John Wiley and Sons, New York, 1979.

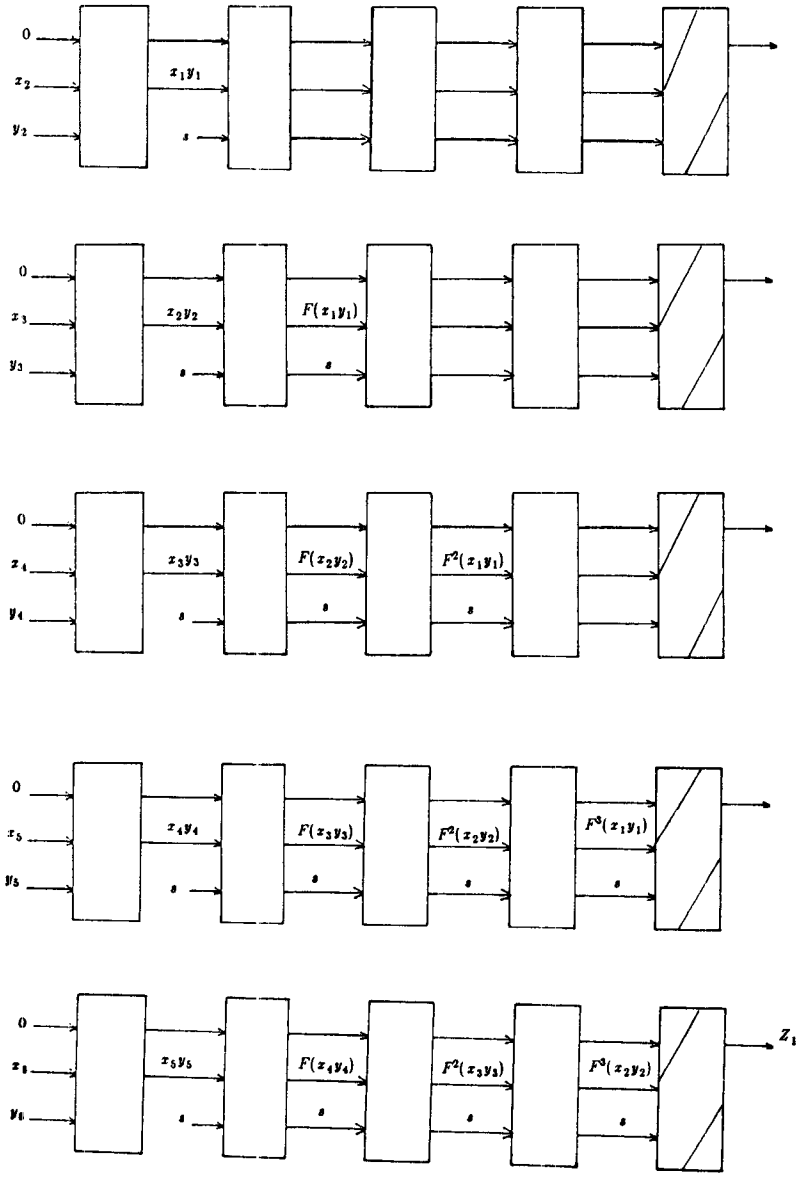

0
$x_2$
$y_2$
$x_1y_1$
$s$
0
$x_3$
$y_3$
$x_2y_2$
$F(x_1y_1)$
$s$
$s$
0
$x_4$
$y_4$
$x_3y_3$
$F(x_2y_2)$
$F^2(x_1y_1)$
$s$
$s$
$s$
0
$x_5$
$y_5$
$x_4y_4$
$F(x_3y_3)$
$F^2(x_2y_2)$
$F^3(x_1y_1)$
$s$
$s$
$s$
0
$x_6$
$y_6$
$x_5y_5$
$F(x_4y_4)$
$F^2(x_3y_3)$
$F^3(x_2y_2)$
$Z_1$
$s$
$s$
$s$

Figure 4.