VLSI DESIGN METHODOLOGIES FOR COMPUTING X mod m

R. Sivakumar [*] *N. J. Dimopoulos*[*]

Technical Report ECE 95-2

Dept of Electrical & Computer Engineering
University of Victoria, Victoria
B.C, CANADA - V8W 3P6

March 1995

Abstract

The feasibility of implementing Residue Number System (RNS) based arithmetic processors has been motivated by the recent developments in microelectronics. In this report, new VLSI architectures are proposed for computing the integer modulo operation X mod m when m is restricted to the values $2^k, 2^k \pm 1$ and composite numbers whose mutually prime factors fall in the above category. Two different design methodologies, namely, the recursive and partition methods are presented and their respective VLSI computational complexities are analyzed. A VLSI chip that computes X mod m where X is a 16-bit number and $m = 3, 5, 6, 7, 9$ and 10 has been implemented using the proposed schemes in $3\mu m$ CMOS technology and typical measurements have yielded a propagation delay of less than 109ns.

[*]Department of Electrical and Computer Engineering, University of Victoria, Victoria

1. Introduction

The Residue Number System (RNS) has attracted considerable attention recently as a potential candidate for designing high performance digital systems brought about by the revolutionary advances in microelectronics. The RNS is a carry-free number system that has the fundamental ability to support a high degree of parallelism, modularity and regularity. These features can be exploited in VLSI to build high speed digital circuits for applications in signal processing [1, 2]. Several algorithms for Fourier transforms, convolution, digital filtering, cryptography and error control coding [3]-[5] require hardware circuits such as adders, multipliers and random number generators that involve arithmetic modulo operations [6]-[8]

In this work, we propose and analyze VLSI architectures for computing the integer modulo operation, $X \bmod m$. The choices $m = 2^k, 2^k - 1$ and $2^k + 1$ are quite popular from the standpoint of computational and hardware efficiency since the arithmetic is simple and amenable for implementation. In addition, the range of m can be expanded using the *Chinese Remainder Theorem* (CRT), which provides the basis for synthesising the modulus of composite numbers. We shall call the hardware circuit that computes $X \bmod m$ as the residue extractor. Section 2 provides the necessary background in residue arithmetic and reviews the well known methods for evaluating $X \bmod \beta^k$, $\beta^k - 1$, $\beta^k + 1$, where β is the radix of the number system. In Sections 3 and 4, two different schemes for designing the residue extractor, namely, the recursive and partition methods are presented. New VLSI architectures are proposed for the two schemes and the respective asymptotic area-time complexities are analyzed in Section 5. Section 6 is devoted to the implementation of a 16-bit residue extractor in $3\mu m$ CMOS technology and typical performance measurements are presented. Finally, the report concludes with a discussion of the merits and demerits of the proposed architectures in relation to other practical implementations.

2. Residue Arithmetic: A Review

Let *X* and *m* be any two integers defined in a number system N of radix β such that $m > 0$. Then we can express $X = mq + r$ where *q* is the quotient and *r* is the remainder such that $0 \leq r < m$ [9]. The remainder *r*, called the *residue* of $X \bmod m$ is designated as $\langle X \rangle_m$. Any positive integer $X \in N$ can be represented as an n-digit tuple $x_{n-1}, x_{n-2}, \ldots, x_1, x_0$ where $n = \left\lceil \log_\beta (X+1) \right\rceil$ such that

$$X = \sum_{i=0}^{n-1} \beta^i x_i \tag{1}$$

2.1. Calculation of $X \bmod \beta^k$

For $k < n-1$, the number X given by eqn. 1 is written as

$$X = \sum_{i=0}^{k-1} \beta^i x_i + \beta^k [x_k + \beta x_{k+1} + \ldots + \beta^{n-1-k} x_{n-1}]$$

Then,

$$\langle X \rangle_{\beta^k} = \langle \sum_{i=0}^{k-1} \beta^i x_i + \beta^k [x_k + \beta x_{k+1} + \ldots \beta^{n-1-k} x_{n-1}] \rangle_{\beta^k} = \sum_{i=0}^{k-1} \beta^i x_i \quad (2)$$

Therefore the number represented by the least significant *k* digits of the representation of *X* gives the residue $\langle X \rangle_{\beta^k}$.

2.2. Calculation of $X \bmod \beta^k - 1$

Before proceeding further, we state the following relations [10] which are important in this analysis.

$$\langle \beta^k \rangle_{\beta^k - 1} = \langle \langle \beta^k - 1 \rangle_{\beta^k - 1} + \langle 1 \rangle_{\beta^k - 1} \rangle_{\beta^k - 1} = 1 \quad (3)$$

By a similar token, we have

$$\langle \beta^{ik} \rangle_{\beta^k - 1} = \langle [\beta^k]^i \rangle_{\beta^k - 1} = \langle [\langle \beta^k \rangle_{\beta^k - 1}]^i \rangle_{\beta^k - 1} = \langle 1^i \rangle_{\beta^k - 1} = 1 \quad (4)$$

where i is some arbitrary integer such that $i \geq 0$.

Let Y_0 denote the number represented by the k least significant digits of X such that $Y_0 = \sum_{j=0}^{k-1} x_j \beta^j$. Similarly, let $Y_1 = \sum_{j=0}^{k-1} x_{k+j} \beta^j$ represent the next k digits of X. Then we can express

$$X = Y_0 + \beta^k Y_1 + \beta^{2k} Y_2 + \ldots \beta^{(\rho-1)k} Y_{\rho-1} = \sum_{i=0}^{\rho-1} Y_i \beta^{ik} \quad (5)$$

where $\rho = \lceil \frac{n}{k} \rceil$ and $Y_i = \sum_{j=0}^{k-1} x_{ik+j} \beta^j$. It therefore follows that

$$\begin{aligned} \langle X \rangle_{\beta^k - 1} &= \langle \sum_{i=0}^{\rho-1} Y_i \beta^{ik} \rangle_{\beta^k - 1} \\ &= \langle \sum_{i=0}^{\rho-1} Y_i \rangle_{\beta^k - 1} \end{aligned} \quad (6)$$

2.3. Calculation of $X \bmod \beta^k + 1$

Making use of the properties

$$\begin{aligned} -1 &\equiv \langle \beta^k \rangle_{\beta^k + 1} \\ (-1)^i &\equiv \langle \beta^{ik} \rangle_{\beta^k + 1} \end{aligned} \quad (7)$$

we obtain

$$\begin{aligned} \langle X \rangle_{\beta^k+1} &= \langle \sum_{i=0}^{\rho-1} Y_i \beta^{ik} \rangle_{\beta^k+1} \\ &= \langle \sum_{i=0}^{\rho-1} (-1)^i Y_i \rangle_{\beta^k+1} \end{aligned} \tag{8}$$

2.4. Calculation of residues of composite numbers

The CRT [10, 11] provides a method for evaluating $\langle X \rangle_m$ when m is a composite number which is not of the form β^k, $\beta^k - 1$, $\beta^k + 1$. If $m = \prod_{i=1}^{p} m_i$ is a product of mutually prime numbers, then the CRT asserts that there is a one-to-one correspondence between an integer $0 \leq X \leq m$ and the p-tuple of its residue classes with respect to m_i, $i = 1, \ldots, p$. Furthermore, given a p-tuple $(x_1, x_2, \ldots, x_p)$ where $0 \leq x_i \leq m_i$, one can find [13] a unique number $0 \leq X \leq m$ such that $x_i = \langle X \rangle_{m_i}$.

We can now define a residue mapping function (RMF) given by $F(x) : P \rightarrow Q$ as follows:

$$\begin{aligned} P &= \{\text{The set of all residue classes generated by } \langle X \rangle_m\} \\ Q &= \{\text{The set of all } p\text{-tuples } (x_1, \ldots, x_p) \text{ where } 0 \leq x_i < m_i \text{ and } 1 \leq i \leq p\} \\ F &= \{(x, \hat{r}) \mid x \in P \text{ and } \hat{r} = (r_1, \ldots, r_p) \in Q \text{ such that } r_i = \langle x \rangle_{m_i}, i = 1, \ldots, p\} \end{aligned} \tag{9}$$

If the factors composing m are of the form $\beta^k \pm 1$ or β^k, then one can compute $\langle X \rangle_{m_i}$ according to the methods outlined in Sections 2.1 to 2.3. The resulting vector of residues can be mapped to the appropriate modulo m class with the help of the RMF function defined by eqn. 9. We shall present examples of this procedure in Section 6.3. The residue mapping function F can be constructed using any of the standard algorithms found in the literature (e.g. Garner's method [13, pp. 253] and implemented in terms of a look-up table stored in a ROM. The size of the ROM needed for the RMF is given as $m \sum_{i=1}^{p} \lceil \log(m_i + 1) \rceil = \mathcal{O}(m \lceil \log(m+1) \rceil)$ bits for the case of the binary number system*. Observe that without the CRT, the size of the ROM needed to store the X mod m function will be $2^n \lceil \log(m+1) \rceil$ bits. In most cases $m \ll n$ and hence the cost of implementing the RMF in a ROM is small compared to a complete table look-up.

3. Recursive Method

In this section, we present a recursive algorithm for computing X mod m when $m = \beta^k - 1$ or $\beta^k + 1$. This algorithm is based on eqns. 6 and 8 presented in Sections 2.2 and 2.3 respectively. Without loss of generality, we focus our discussion for the the case $m = \beta^k - 1$. The original problem, that of evaluating X mod $\beta^k - 1$ is reduced to finding $\langle \sum_{i=0}^{\rho-1} Y_i \rangle_{\beta^k - 1}$. If now the

*Unless otherwise specified, $\log n$ denotes $\log_2 n$.

representation of $\sum_{i=0}^{\rho-1} Y_i$ needs less digits as compared to that of the original number X, then we can recursively apply the same procedure, each time computing† $Z^{(j)} = \sum_{i=0}^{\rho^{(j-1)}-1} Y_i^{(j-1)}$, $j = 1, \ldots, \gamma$ where γ is an integer bounded from above by $n - k$ such that at the γ^{th} iteration $Z^{(\gamma)} = \sum_{i=0}^{\rho^{(\gamma-1)}-1} Y_i^{(\gamma-1)} \leq \beta^k - 1$ and the algorithm terminates.

Since $Z^{(1)} \leq \lceil \frac{n}{k} \rceil \beta^k - \lceil \frac{n}{k} \rceil$, the upper bound on the representation size of $Z^{(1)}$ is given by $n^{(1)} = \left\lceil \log_\beta \rho^{(0)} + k \right\rceil = \left\lceil \log_\beta \lceil \frac{n}{k} \rceil + k \right\rceil$. Observe now that $n^{(1)} \leq n$ for $n \geq k$. Indeed $n^{(1)} = \left\lceil \log_\beta \lceil \frac{n}{k} \rceil + k \right\rceil = \left\lceil \log_\beta \lceil \frac{n}{k} \rceil \right\rceil + k = \left\lceil \log_\beta \frac{n}{k} \right\rceil + k \leq \log_\beta \frac{n}{k} + k + 1$ ([12]). But $f(n) = n - \log_\beta \frac{n}{k} - k - 1 = n - \log_\beta n + \log_\beta k - k - 1$ is a monotonically increasing function of n and since $f(k+2) = k + 2 - \log_\beta (k+2) + \log_\beta k - k - 1 = 1 - \log_\beta (\frac{k+2}{k}) = 1 - \log_\beta (1 + \frac{k}{2}) \geq 0$, $\forall k \geq 2$. Then $n^{(1)} < n$ for $k > 1$ and $n \geq k + 2$.

Case 1: $k = 1$

$n^{(1)} = \lceil \log_\beta n \rceil + 1 \leq n, \forall n \geq 1.$

Case 2: $n = k$

$n^{(1)} = \lceil \log_\beta \frac{k}{k} \rceil + k = k = n.$

Case 3: $n = k + 1$

$n^{(1)} = \lceil \log_\beta (1 + \frac{1}{k}) \rceil + k \leq k + 1 = n.$

Note that $n^{(1)} < n$ for $n \geq k + 2$ and $k > 1$. Thus, the number of digits of the sum at the end of each iteration is reduced and the algorithm eventually terminates. The number of k-digit numbers required for addition in the j^{th} iteration is given by

$$\rho^{(j-1)} = \left\lceil \frac{n^{(j-1)}}{k} \right\rceil \tag{10}$$

where $1 \leq j \leq \gamma - 1$. If $n^{(j)}$ is the upper bound of the digits needed to represent the resulting sum $Z^{(j)}$ at the end of the j^{th} iteration, then $n^{(j)} \leq \lceil \log_\beta \rho^{(j-1)} + k \rceil = \lceil \log_\beta \frac{n^{(j-1)}}{k} \rceil + k$. At the end of the $(\gamma - 1)^{th}$ iteration, $n^{(\gamma-1)} = k + 1$. Therefore, in the next iteration, a maximum of two addition steps are sufficient to determine $Z^{(\gamma)}$. The second addition is necessary to account for any overflow that may arise from the first summation. A decoding logic is also incorporated in the final stage $(\gamma + 1)$ to decipher the case $\langle \beta^k - 1 \rangle_{\beta^k - 1} = 0$ which will be called the *zero condition*.

The recurrence relation outlined above is unfolded into $\gamma + 1$ levels each of which computes the sums outlined in eqns. 6 and 8 except the last level and the time complexity of the algorithm can be given as

$$T(n) = \begin{cases} T(\lceil \log_\beta \lceil \frac{n}{k} \rceil + k \rceil) + t(\lceil \frac{n}{k} \rceil) & \text{if } n > k + 1 \\ 2t(2) + t'(k) & \text{if } n = k + 1 \\ t'(k) & \text{if } n < k + 1 \end{cases} \tag{11}$$

† Initially $\rho^{(0)} = \rho = \lceil \frac{n}{k} \rceil$ and $Y_i^{(0)} = Y_i$.

where $t(n)$ is the time taken to add n k-digit numbers while $t'(k)$ is the time required to decode the zero condition for a k-digit number.

For the case $m = \beta^k + 1$, the procedure is similar but for alternating additions and subtractions in deriving Z at each stage.

To illustrate the algorithm with an example, let us consider the ternary number system ($\beta = 3$). Furthermore[‡], let $X = 021021|_3$, $k = 2$ and $m = 8|_{10} = 3^2 - 1$. Then, we can calculate $n = 6|_{10}$ and $\rho = 3|_{10}$. Since m is of the form $\beta^k - 1$, X mod m can be computed as follows:

Step 1: $Y_1^{(0)} = 21|_3$; $Y_2^{(0)} = 10|_3$; $Y_3^{(0)} = 02|_3$; $n^{(1)} = 3$ and $Z^{(1)} = Y_1^{(0)} + Y_2^{(0)} + Y_3^{(0)} = 110|_3$.

Step 2: $Y_1^{(1)} = 10|_3$; $Y_2^{(1)} = 1|_3$; $n^{(2)} = 2$ and $Z^{(2)} = Y_1^{(1)} + Y_2^{(1)} = 11|_3$

Thus $\langle 021021|_3 \rangle_8 = \langle 110|_3 \rangle_8 = \langle 11|_3 \rangle_8 = \langle 4 \rangle_8 = 4$

A generalized algorithm for computing X mod $\beta^k - 1, \beta^k + 1$ based on the above discussion is given in Fig. 1. Procedure Residue is invoked initially with the call Residue($sign_X, |X|, n, \beta, k, m$) where $sign_X = \pm 1$. In Steps 1 and 2, the sum Z is computed. To ensure the correctness of the result, the sign of Z is updated in Steps 3 and 4 based on its current value, $sign_Z$ and previous value $sign_X$. Step 5 checks whether the last stage (γ) has been reached at which point $n = k + 1$. In that case, one more addition is performed in Lines(16)-(17) and the variables, $\hat{n}$, Z and $sign_Z$ are updated in Lines(19)-(21) to ensure proper termination. If $n > k + 1$, the upper bound on the size of Z is calculated in Line (24) and the procedure is recursively invoked in Line (26). Lines (28)-(30) in Step 6 are derived based on the result:

$$\langle Z \rangle_m = \begin{cases} 0 & \text{if } |Z| = m \text{ or } |Z| = 0 \\ m - |Z| & \text{if } Z < 0 \text{ and } |Z| < m \\ Z & \text{if } Z > 0 \text{ and } Z < m \end{cases}$$

4. Partition Method

A residue extractor designed according to the discussion in Section 3 above can handle arguments with a fixed number of digits. One can use several such extractors to realize a modular structure capable of handling larger numbers. The technique which we call the partition method, separates the argument into portions that can be handled individually by the original residue extractor and finally combines the results. Specifically, the given number X is partitioned into λ sub-blocks of s-digits each and the residues of each of the λ sub-blocks are evaluated using the recursive approach. The outputs of these sub-blocks are processed by a tree of Residue Adders (RA) which compute the residue by reduction through an addition modulo m operation. Fig. 2 shows the general organization of this structure. We proceed as follows

$$\langle X \rangle_m = \langle V_0 + \beta^s V_1 + \ldots + \beta^{is} V_i + \ldots + \beta^{s(\lambda-1)} V_{\lambda-1} \rangle_m, \quad \text{for } 0 \leq i \leq \lambda - 1$$

[‡]We use the notation $|_\beta$ to denote a number expressed in base β. If this is omitted, then it is assumed that the number is expressed in base 10.

(1) **Procedure** Residue($sign_X, X, n, \beta, k, m$):
(2) Input : $sign_X$, X, n, β, k, m
(3) Output : X mod m
(4) **begin**
(5) Step 1. Represent the n-digit number X as a weighted sum of powers of β^k as per eqn. 5;
(6) Step 2. **Case**(m) **begin** /* Branch depending on the value of m */
(7) $\beta^k + 1$: Compute $Z = \sum_{i=0}^{\rho-1}(-1)^i Y_i$
(8) $\beta^k - 1$: Compute $Z = \sum_{i=0}^{\rho-1} Y_i$
(9) **end**
(10) where $Y_i = \sum_{j=0}^{k-1} x_{ik+j}\beta^j$ and $\rho = \lceil \frac{n}{k} \rceil$.
(11) Step 3. $sign_Z =$ sign(Z) /* Determine the sign of Z */
(12) Step 4. $sign_Z = sign_Z * sign_X$ /* Update $sign_Z$ */
(13) Step 5. **if** ($n = k + 1$) **then begin** /* Second addition at the γ^{th} stage */
(14) Let $|Z|$ be represented as $Y_0' + Y_1'$ where $Y_0' = \sum_{j=0}^{k-1} x_j'\beta^j$ and $Y_1' = x_k'$.
(15) **Case**(m) **begin**
(16) $\beta^k + 1$: Calculate $Z' = Y_0' - Y_1'$
(17) $\beta^k - 1$: Calculate $Z' = Y_0' + Y_1'$
(18) **end** /* of Case */
(19) $\hat{n} = k$
(20) $sign_Z =$ sign(Z) $* sign(Z')$
(21) $Z = |Z'|$
(22) **end**
(23) **else**
(24) $\hat{n} = \left\lceil \log_\beta \lceil \frac{n}{k} \rceil + k \right\rceil$ /* Calculate size of Z */
(25) Step 6. **if** $\hat{n} > k$ **then**
(26) **call** Residue($sign_Z, |Z|, \hat{n}, \beta, k, m$)
(27) **else**
(28) **if** $|Z| = m$ **or** $|Z| = 0$ **then** $residue = 0$ /* Zero Condition */
(29) **else begin**
(30) **if** $sign_Z < 0$ **then** $residue = m - |Z|$ **else** $residue = |Z|$
(31) **return**(residue)
(32) **end**
(33) **end** /* of Procedure Residue */

Figure 1 *A Recursive Algorithm for computing* X mod $\beta^k - 1, \beta^k + 1$

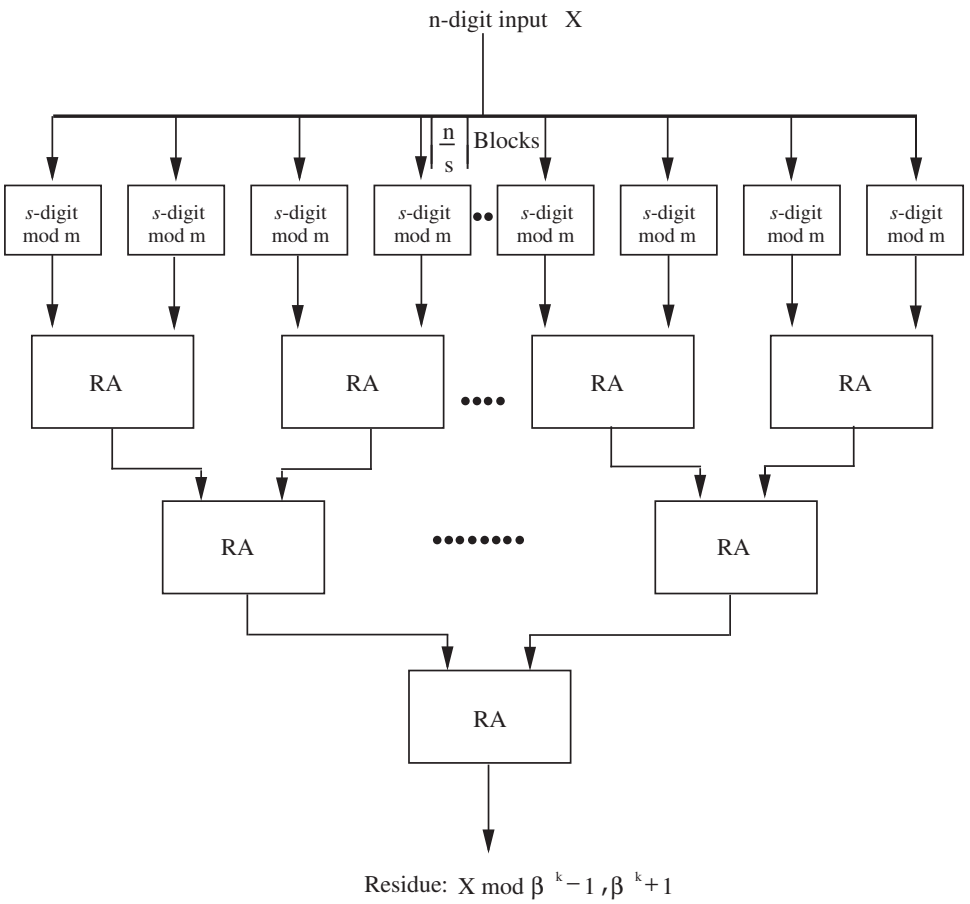

Figure 2 *Logical structure of the partition method for computing* $X \mod \beta^k - 1, \beta^k + 1$.

where $\lambda = \lceil \frac{n}{s} \rceil$ and $V_i = \sum_{j=0}^{s-1} x_{is+j} \beta^j$.

From eqns. 4 and 7, it follows that if $\eta = \frac{s}{k}$ is an integer then $\langle \beta^s \rangle_m = \langle (\beta^k)^{\frac{s}{k}} \rangle_m = \langle (\beta^k)^\eta \rangle_m = 1^\eta$ and $\langle \beta^{si} \rangle_m = 1^{\eta i}$ when $m = \beta^k - 1$. By a similar token, $\langle \beta^s \rangle_m = (-1)^\eta$ and $\langle \beta^{si} \rangle_m = (-1)^{\eta i}$ for $m = \beta^k + 1$. Hence in the partition method, the number of digits s is chosen such that s is a multiple of k and this simplifies the computation as given below:

Case 1: $m = \beta^k - 1$

$$\begin{aligned} \langle X \rangle_m &= \langle \langle \langle V_0 \rangle_m + \langle V_1 \rangle_m \rangle_m + \ldots + \langle V_{\lambda-1} \rangle_m \rangle_m \\ &= \langle \sum_{i=0}^{\lambda-1} \langle V_i \rangle_m \rangle_m \end{aligned} \tag{12}$$

Case 2: $m = \beta^k + 1$

$$\langle X \rangle_m = \langle \langle V_0 \rangle_m + (-1)^\eta \langle V_1 \rangle_m + \ldots + (-1)^{\eta(\lambda-1)} \langle V_{\lambda-1} \rangle_m \rangle_m$$

$$= \langle \sum_{i=0}^{\lambda-1} (-1)^{\eta i} \langle V_i \rangle_m \rangle_m \tag{13}$$

Thus X mod $\beta^k - 1, \beta^k + 1$ can be computed as a modulo m summation of the residues obtained from each s-digit block. The binary tree arrangement of the residue adders permit computation to proceed in parallel in $\lceil \log_2 \lambda \rceil$ steps. While considering the limiting cases, observe that when $s = n$, there is exactly one X mod m block and the partition method is identical to the recursive method of Section 3. On the other hand for $s = k$, the s-digit modulo m units in the representation are not used and the partition method reduces to a circuit consisting of $\lambda - 1$ residue adders arranged as a binary tree [14] each adding/subtracting two k-bit numbers as the case may be and evaluating the residue with respect to m. This circuit is a special case of the partition method where the modulus is evaluated by reduction through a cluster of residue adders.

5. Complexity Analysis

In this section, the proposed structures obtained from the recursive and partition methods are evaluated as an asymptotic VLSI design for the case of the binary number system ($\beta = 2$) which is a tenable medium for implementation. The complexity measures for the area *A* and the propagation delay time T of the circuits are derived based on the assumptions given in [15, pp. 29-38, 137-138].

5.1. Recursive Method

5.1.1. Area

In Section 3, it was shown that the evaluation of X mod m can be accomplished by recursively adding $\lceil \frac{n^{(i)}}{k} \rceil$ k-bit factors, $i = 1, \ldots, \gamma - 1$ and that the representative length of the sum, namely $n^{(i)}$ reduces as given by eqn. 10. The residue can therefore be computed by repetitive summation of k-bit words which is akin to the partial product reduction in the multipliers problem [16]. The hardware complexity, A_{rm} is given as

$$\begin{aligned} A_{rm} &= \sum_{i=1}^{\gamma} H(\rho^{(i-1)}) = H\left(\left\lceil \frac{n^{(0)}}{k} \right\rceil\right) + H\left(\left\lceil \frac{n^{(1)}}{k} \right\rceil\right) + \ldots + H\left(\left\lceil \frac{n^{(\gamma-2)}}{k} \right\rceil\right) \\ &\quad + \ldots + H(2) \\ &= H\left(\left\lceil \frac{n}{k} \right\rceil\right) + H\left(\left\lceil \frac{\log \lceil \frac{n}{k} \rceil}{k} + k \right\rceil\right) + H\left(\left\lceil \frac{\log \left\lceil \frac{\log \lceil \frac{n}{k} \rceil}{k} + k \right\rceil}{k} + k \right\rceil\right) + \ldots \\ &\quad + \ldots + H(2) \end{aligned} \tag{14}$$

where $H(\rho^{(i-1)})$ is the area complexity for adding $\rho^{(i-1)}$ k-bit numbers for $i = 1, \ldots, \gamma - 1$ while $H(2)$ represents the complexity for adding two k-bit numbers in the γ^{th} stage. Schematically, the

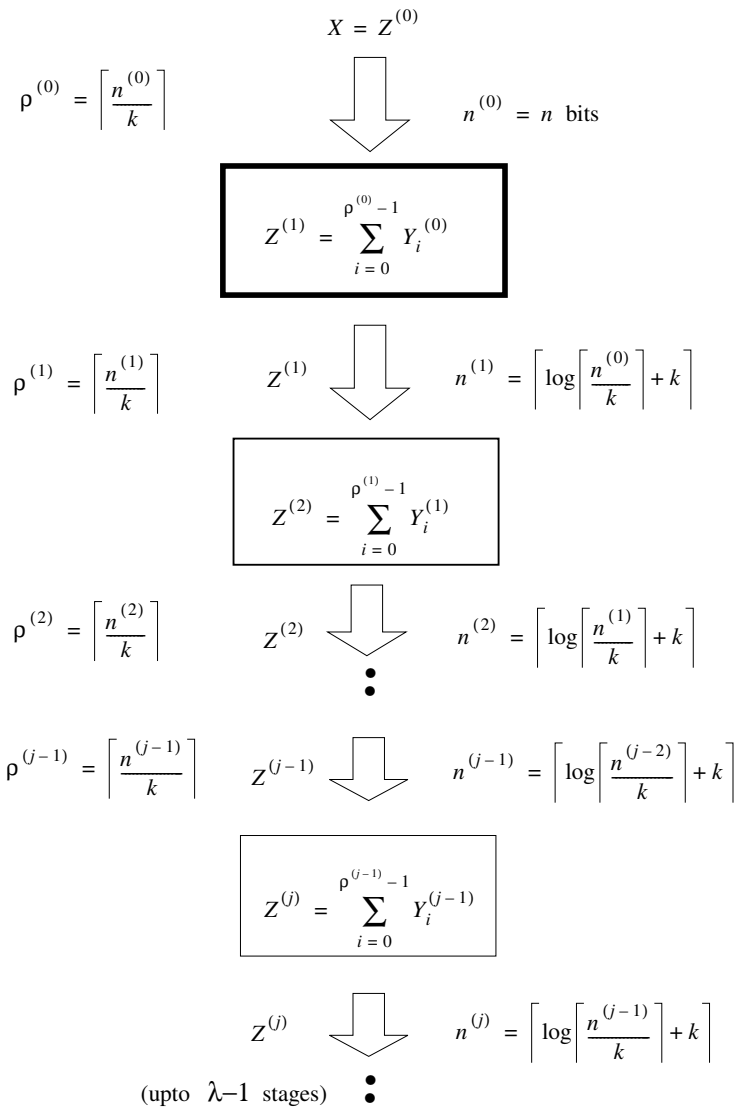

Figure 3 *Schematic diagram of the Recursive method.*

recursive method is unfolded as depicted in Fig. 3.

From the above eqn., it can be seen that the recursive method is dictated by the multi-operand addition in the first step of the algorithm. According to the parallel counter schemes advocated in [17, 18], the hardware can be synthesised by using k-bit adders as the major computational element. Let the area occupied by a k-bit adder [15] be denoted by $A(k) = \mathcal{O}(k \log k)$. Then the complexity of adding $\lceil \frac{n}{k} \rceil$ words of k-bits can be derived as follows.

For the sake of simplicity, let us assume that $\lceil \frac{n}{k} \rceil$ is a power of two. If the adders are arranged in the form of a binary tree [17] for computing $\lceil \frac{n}{k} \rceil$ k-bits words then the first level consists of $\frac{\lceil \frac{n}{k} \rceil}{2}$ k-bit adders. The second level comprises of $\frac{\lceil \frac{n}{k} \rceil}{4}$ $(k+1)$-bit adders and so on till $\log \lceil \frac{n}{k} \rceil$ stages at which point, there is one adder of complexity $(k + \log \lceil \frac{n}{k} \rceil)$-bits. A $(k+i)$-bit adder block ($i = 2, \ldots, \log \lceil \frac{n}{k} \rceil - 1$) can be decomposed into a k-bit adder and an i-bit adder where such that the carry output of the k-bit adder is chained to the i-bit adder. Then the area complexity, $H\left(\lceil \frac{n}{k} \rceil\right)$, can be determined by computing the complexity of the k-bit adder blocks and the extra adders required for accumulating the overflow bits at each level.

$$\begin{aligned}
H\left(\left\lceil \frac{n}{k} \right\rceil\right) &= \sum_{i=1}^{\log \lceil \frac{n}{k} \rceil} \frac{\lceil \frac{n}{k} \rceil}{2^i} A(k) + \sum_{i=2}^{\log \lceil \frac{n}{k} \rceil} \frac{\lceil \frac{n}{k} \rceil \, A(i-1)}{2^i} \\
&= \left(\left\lceil \frac{n}{k} \right\rceil - 1\right) k \log k + \sum_{i=2}^{\log \lceil \frac{n}{k} \rceil} \frac{\lceil \frac{n}{k} \rceil (i-1) \log(i-1)}{2^i} \\
&= \left(\left\lceil \frac{n}{k} \right\rceil - 1\right) k \log k + \sum_{i=1}^{\log \lceil \frac{n}{k} \rceil - 1} \frac{\lceil \frac{n}{k} \rceil i \log i}{2^{i+1}} \\
&= \left(\left\lceil \frac{n}{k} \right\rceil - 1\right) k \log k + \frac{\lceil \frac{n}{k} \rceil}{2} \sum_{i=1}^{\log \lceil \frac{n}{k} \rceil - 1} \frac{i \log i}{2^i}
\end{aligned}$$

Since $x \log x < 2^x$ for integer values of $x \geq 1$, the total area is bounded from above by

$$\begin{aligned}
H\left(\left\lceil \frac{n}{k} \right\rceil\right) &< \left(\left\lceil \frac{n}{k} \right\rceil - 1\right) k \log k + \frac{\lceil \frac{n}{k} \rceil}{2} \sum_{i=1}^{\log \lceil \frac{n}{k} \rceil - 1} 1 \\
&= \left(\left\lceil \frac{n}{k} \right\rceil - 1\right) k \log k + \frac{\lceil \frac{n}{k} \rceil \left(\log \lceil \frac{n}{k} \rceil - 1\right)}{2} \qquad (15) \\
&= \mathcal{O}\left(\max\left\{\left\lceil \frac{n}{k} \right\rceil k \log k, \left\lceil \frac{n}{k} \right\rceil \log \left\lceil \frac{n}{k} \right\rceil\right\}\right) \\
&= \mathcal{O}(n \log n) \qquad (16)
\end{aligned}$$

When $n \gg k$, the asymptotic area complexity of the recursive method can be obtained by substituting eqn. 16 in eqn. 14.

$$\begin{aligned}
A_{rm} &= \mathcal{O}(n \log n) + \mathcal{O}(\log n \log \log n) + \mathcal{O}(\log \log n \log \log \log n) + \ldots \\
&\quad \ldots + \mathcal{O}(2k \log k) \qquad (17)
\end{aligned}$$

$$\begin{aligned} &\leq c\left(n\log n + \log n \log\log n + \log n \log\log\log n + \ldots + 2k\log k\right)) \\ &\leq c\log n(n + \log\log n + \log\log\log n + \ldots) + 2ck\log k \end{aligned} \tag{18}$$

where c is a given constant. Note that in eqn. 17 the term $\mathcal{O}(2k\log k)$ refers to the area complexity of the two k-bit adders in the γ^{th} stage.

Each successive term in the eqn. 18 progressively diminishes for $n \geq 1$. Hence, eqn. 18 can be bounded by a converging geometric progression such that

$$A_{rm} \leq c\log n\sum_{i=0}^{\gamma-1} n\alpha^i + 2ck\log k = cn\log n\frac{(1-\alpha^\gamma)}{(1-\alpha)} + 2ck\log k \quad \text{if } \alpha \leq 1 \tag{19}$$

Observe that the ratio of the second and the first terms of eqn. 18 is given by $\alpha = \frac{\log\log n}{n}$ whose maximum can be derived by computing

$$\frac{d\alpha}{dn} = \frac{1}{\log_e 2}\left(\frac{1}{\log_e\left(\frac{n}{\log_e 2}\right)} - \log\log\left(\frac{n}{\log_e 2}\right)\right) = 0$$

The numerical solution of the above equation yields $\alpha^* = 0.250110$. Therefore, substituting α^* in eqn. 19, we obtain

$$\begin{aligned} A_{rm} &\leq cn\log n\frac{(1-\alpha^\gamma)}{(1-\alpha)} + 2ck\log k \leq \frac{cn\log n}{1-\alpha} \\ &= 1.3335cn\log n + 2ck\log k \\ &= \mathcal{O}(n\log n) \end{aligned} \tag{20}$$

For the case $k = \mathcal{O}(n)$, the area complexity measure of the recursive method is similar to that of eqn. 20 since the complexity of the adder is $\mathcal{O}(n\log n)$.

5.1.2. Time

The time complexity, T_{rm}, of the recursive method is given by

$$\begin{aligned} T_{rm} &= \sum_{i=1}^{\gamma} t(\rho^{(i)}) \\ &= t\left(\left\lceil\frac{n}{k}\right\rceil\right) + t\left(\left\lceil\frac{\log\lceil\frac{n}{k}\rceil}{k} + k\right\rceil\right) + t\left(\left\lceil\frac{\log\left\lceil\frac{\log\lceil\frac{n}{k}\rceil}{k} + k\right\rceil}{k} + k\right\rceil\right) + \ldots \\ &\qquad + \ldots + t(2) \end{aligned} \tag{21}$$

Let the time complexity of a k-bit adder [15] be given as $\tau(k) = \mathcal{O}(\log k)$. With the assumptions made in Section 5.1.1, the time required to add $\lceil\frac{n}{k}\rceil$ k-bit words can be calculated by considering the delay of the k-bit adder and the extra adders at each level of the binary tree.

Therefore

$$\begin{aligned} t\left(\left\lceil\frac{n}{k}\right\rceil\right) &= \sum_{i=1}^{\log\left\lceil\frac{n}{k}\right\rceil}\tau(k)+\sum_{j=1}^{\log\left\lceil\frac{n}{k}\right\rceil-1}\tau(j) \\ &= \log\left\lceil\frac{n}{k}\right\rceil\log k+\sum_{j=1}^{\log\left\lceil\frac{n}{k}\right\rceil-1}\log j \\ &\leq \log\left\lceil\frac{n}{k}\right\rceil\log k+\left(\log\left\lceil\frac{n}{k}\right\rceil-1\right)\log\left(\log\left\lceil\frac{n}{k}\right\rceil-1\right) \\ &= \begin{cases}\mathcal{O}(\log n) & \text{if } k=\mathcal{O}(n)\\ \mathcal{O}(\log n\log\log n) & \text{if } n\gg k\end{cases} \end{aligned} \tag{22}$$

Substituting eqn. 22 in eqn. 21, we obtain the time complexity of the recursive method as

$$\begin{aligned} T_{rm} &\leq c'(\log n+\log\log n+\log\log\log n+\ldots 2\log k) \\ &= \mathcal{O}(\log n) \quad \text{if } k=\mathcal{O}(n) \qquad (23)\\ &\leq c'(\log n\log\log n+\log\log n\log\log\log n+\ldots+2\log k) \\ &= \mathcal{O}(\log n\log\log n) \quad \text{if } n\gg k \qquad (24) \end{aligned}$$

where c' is a constant.

5.2. Partition Method

5.2.1. Area

As mentioned previously, X mod m is evaluated for each s-bit block using the recursive method and the output is processed through a tree of residue adders. The residue adders are versions of conventional adders [19] that perform the addition modulo m operation $\langle\langle X_1\rangle_m \pm \langle X_2\rangle_m\rangle_m$. For example, when $m=2^k-1$, the residue adder computes the sum according to the following conditions.

1. If $\langle X_1\rangle_{2^k-1}+\langle X_2\rangle_{2^k-1}<2^k-1$, then $\langle X_1\rangle_{2^k-1}+\langle X_2\rangle_{2^k-1}$ yields the desired result.
2. If $\langle X_1\rangle_{2^k-1}+\langle X_2\rangle_{2^k-1}=2^k-1$, then we have a string of k 1's and the result is decoded to be zero since $\langle 2^k-1\rangle_{2^k-1}=0$.
3. If $\langle X_1\rangle_{2^k-1}+\langle X_2\rangle_{2^k-1}>2^k-1$, then the addition will result in a $k+1$ bit number, and we have

$$\begin{aligned} \langle\langle X_1\rangle_{2^k-1}+\langle X_2\rangle\rangle_{2^k-1} &= \langle 2^k+\langle\langle X_1\rangle_{2^k-1}+\langle X_2\rangle_{2^k-1}\rangle_{2^k}\rangle_{2^k-1} \\ &= 1+\langle\langle X_1\rangle_{2^k-1}+\langle X_2\rangle_{2^k-1}\rangle_{2^k} \end{aligned}$$

The required solution is obtained by adding the carry bit to the least significant k-bits after the first addition is performed. In effect, two levels of addition are required. The residue adder

for $m = 2^k - 1$ is composed of two adders, the first one for adding the two k-bit numbers and the second for absorbing the resultant carry, if any, to the previous sum. The adders can be constructed to have an area and time complexity [20] of $\mathcal{O}(k \log k)$ and $\mathcal{O}(\log k)$ respectively while the decoding circuit for the zero condition can be assumed to have $\mathcal{O}(\log k)$ delay also. The above circuit can be adapted to perform 1's complement subtraction, for instance, when $m = 2^k + 1$ with comparable complexity.

The area of the circuit using the partition approach is composed of two parts, namely, the area occupied by the s-bit stages A_s and the residue adder tree A_t. The area for the s-bit stages comprising of $\lceil \frac{n}{s} \rceil$ blocks can be obtained by substituting s/k for n in eqn. 20.

$$\begin{aligned} A_s &= \left\lceil \frac{n}{s} \right\rceil \mathcal{O}\left(\frac{s}{k} \log \frac{s}{k}\right) \\ &= \mathcal{O}\left(\left(\frac{n}{s}\right)\left(\frac{s}{k}\right) \log \frac{s}{k}\right) = \mathcal{O}\left(\frac{n}{k} \log \frac{s}{k}\right) \end{aligned} \tag{25}$$

The area for the residue adder tree is given by

$$\begin{aligned} A_t &= \left(\sum_{i=1}^{\log \lceil \frac{n}{s} \rceil} 2^{i-1} \right) A^*(k) \\ &= 2\left(\left\lceil \frac{n}{s} \right\rceil - 1\right) A(k) \end{aligned} \tag{26}$$

where $A^*(k) = 2A(k)$ accounting for the two levels of addition in the residue adder.

The total area A_p can be obtained from eqns. 25 and 26 as follows

$$\begin{aligned} A_p &= \mathcal{O}\left(\frac{n}{k} \log\left(\frac{s}{k}\right)\right) + 2\left(\left\lceil \frac{n}{s} \right\rceil - 1\right) \mathcal{O}(k \log k) \\ &= \mathcal{O}\left(\frac{n}{k} \log \frac{s}{k}\right) + \mathcal{O}\left(\left(\frac{n}{s}\right) k \log k\right) \end{aligned}$$

The asymptotic area complexities of the partition method is then given by

$$A_p = \begin{cases} \mathcal{O}(n) & \text{if } n \gg s, k \\ \mathcal{O}(n \log n) & \text{if } s, k = \mathcal{O}(n) \end{cases} \tag{27}$$

5.2.2. Time

The time complexity for the partition method is dictated by the delay of the s-bit blocks and the delay of the residue adder tree. Substituting the upper bound of the delay from eqn. 24, the time complexity T_s for the s-bit blocks is given as

$$T_s = \mathcal{O}\left(\log\left(\frac{s}{k}\right) \log\log\left(\frac{s}{k}\right)\right) \tag{28}$$

The residue adder tree has a delay given by

$$T_r = \log\left\lceil \frac{n}{s} \right\rceil \tau^*(k) \tag{29}$$

where $\tau^*(k) = 2\tau(k)$.

Hence the time complexity, T_p, of the partition method is given by combining eqns. 28 and 29:

$$T_p = \mathcal{O}\left(\log\left(\frac{s}{k}\right)\log\log\left(\frac{s}{k}\right)\right) + 2\log\left\lceil\frac{n}{s}\right\rceil\mathcal{O}(\log k) \tag{30}$$

The asymptotic time complexities can then be approximated as

$$T_p = \begin{cases} \mathcal{O}(\log n \log\log n) & \text{if } s = \mathcal{O}(n) \\ \mathcal{O}(\log n^2) & \text{if } n \gg s, k \\ \mathcal{O}(\log n) & \text{if } s, k = \mathcal{O}(n) \end{cases} \tag{31}$$

From the above analysis, it can be observed that the area complexity for the recursive approach is $\mathcal{O}(n \log n)$ while for the partition method it is in the range $[\mathcal{O}(n) \ldots \mathcal{O}(n \log n)]$. On the other hand, the time complexities of the two methods vary between $[\mathcal{O}(\log n) \ldots \mathcal{O}(\log n \log\log n)]$. It can be concluded that the partition approach has an area-time complexity that is better than or equal to the recursive approach and the choice of the partition size s will dictate the performance to cost benefits.

6. Implementation

A 16-bit residue-extractor for positive integers with moduli $m = 3, 5, 6, 7, 9$ and 10 has been implemented in 3μ CMOS technology using the recursive and partition methods discussed here. In order to illustrate the methodology behind the two proposed architectures, we present the basic block diagrams and discuss the implementation issues for the cases $m = 5$, 6 and 7.

6.1. X mod 5

In this case, observe that the modulo is of the type $2^k - 1$ with $k = 2$. The implementation of $\langle X \rangle_5$ is based on the partition method. The number X whose size is 16-bits (excluding the sign), is partitioned into two blocks V_0 and V_1 such that $V_0 = \sum_{i=0}^{7} 2^i x_i$ and $V_1 = \sum_{i=0}^{7} 2^i x_{i+8}$ are two 8-bit numbers. Here $s = 8$, $\lambda = 2$, $\eta = 4$ and $\rho = 2$. Applying eqn. 13, we obtain $\langle X \rangle_5 = \langle \langle V_0 \rangle_5 + \langle V_1 \rangle_5 \rangle_5$ The output from the two blocks are exported to a residue adder to obtain the final result $\langle X \rangle_5$.

The recursive method is used to compute the residues of each of the two partitions, namely, $\langle V_0 \rangle_5$ and $\langle V_1 \rangle_5$. These terms are calculated using eqn. 8 such that

$$\langle V_0 \rangle_5 = \langle Y_0 - Y_1 + Y_2 - Y_3 \rangle_5$$
$$\langle V_1 \rangle_5 = \langle Y_4 - Y_5 + Y_6 - Y_7 \rangle_5$$

where $Y_i = \sum_{j=0}^{1} x_{2i+j} 2^j$, $i \in \{0, \ldots, 7\}$.

A block diagram for computing $\langle V_0 \rangle_5$ is illustrated in Fig. 4. The key components of this schematic include 2-bit subtractor/adder modules, register interfaces, a sign update module, a zero

Table 1 $F(x) : P \rightarrow Q$

P	Q
$\langle X \rangle_6$	$\langle X \rangle_2 \langle X \rangle_3$
0	00
1	11
2	02
3	10
4	01
5	12

condition detect and negative modulus correction circuitry corresponding to procedure *Residue* discussed earlier. The grouping of bits for array reduction is shown in the top right of Fig. 4. This design has 4 stages with each stage having a delay approximately equal to that of a 2-bit adder/subtractor unit. The implemented circuit is non-pipelined. However it can be easily pipelined with the addition of register modules at every stage.

6.2. X mod 7

The modulo is of type $2^k - 1$ with $k = 3$ and $\langle X \rangle_7 = \langle Y_0 + Y_1 + Y_2 + Y_3 + Y_4 + Y_5 \rangle_7$ where $Y_i = \sum_{j=0}^{2} x_{3i+j} 2^j$, $i \in \{0, \ldots, 4\}$ and $Y_5 = x_{15}$. Fig. 5 pictorially illustrates the organization of the various blocks of the X mod 7 circuit which uses the recursive approach and consists of 6 stages.

6.3. X mod 6

We follow the technique presented in Section 2.4 to compute X mod 6 using the RMF. The number 6 can be expressed as the product of two relatively prime factors, m_1 and m_2 where $m_1 = 2$ and $m_2 = 3$. Therefore, $\langle X \rangle_6$ can be computed if $\langle X \rangle_2$ and $\langle X \rangle_3$ are known. From eqn.(2) and $k = 1$, we obtain $\langle X \rangle_2 = x_0$ where x_0 is the least significant bit. $\langle X \rangle_3$ (where 3 is expressed in the form $2^k - 1 = 2^2 - 1$) can be evaluated using either of the partition or recursive methods. We chose the partition method with two 4-bit partitions to extract $\langle X \rangle_3$.

Table 1 gives the RMF defined by eqn. 9 for synthesizing $\langle X \rangle_6$ and the required boolean logic can be derived easily from $\langle X \rangle_2$ and $\langle X \rangle_3$. Likewise, $\langle X \rangle_{10}$ can be synthesised from $\langle X \rangle_2$ and $\langle X \rangle_5$ through a ROM.

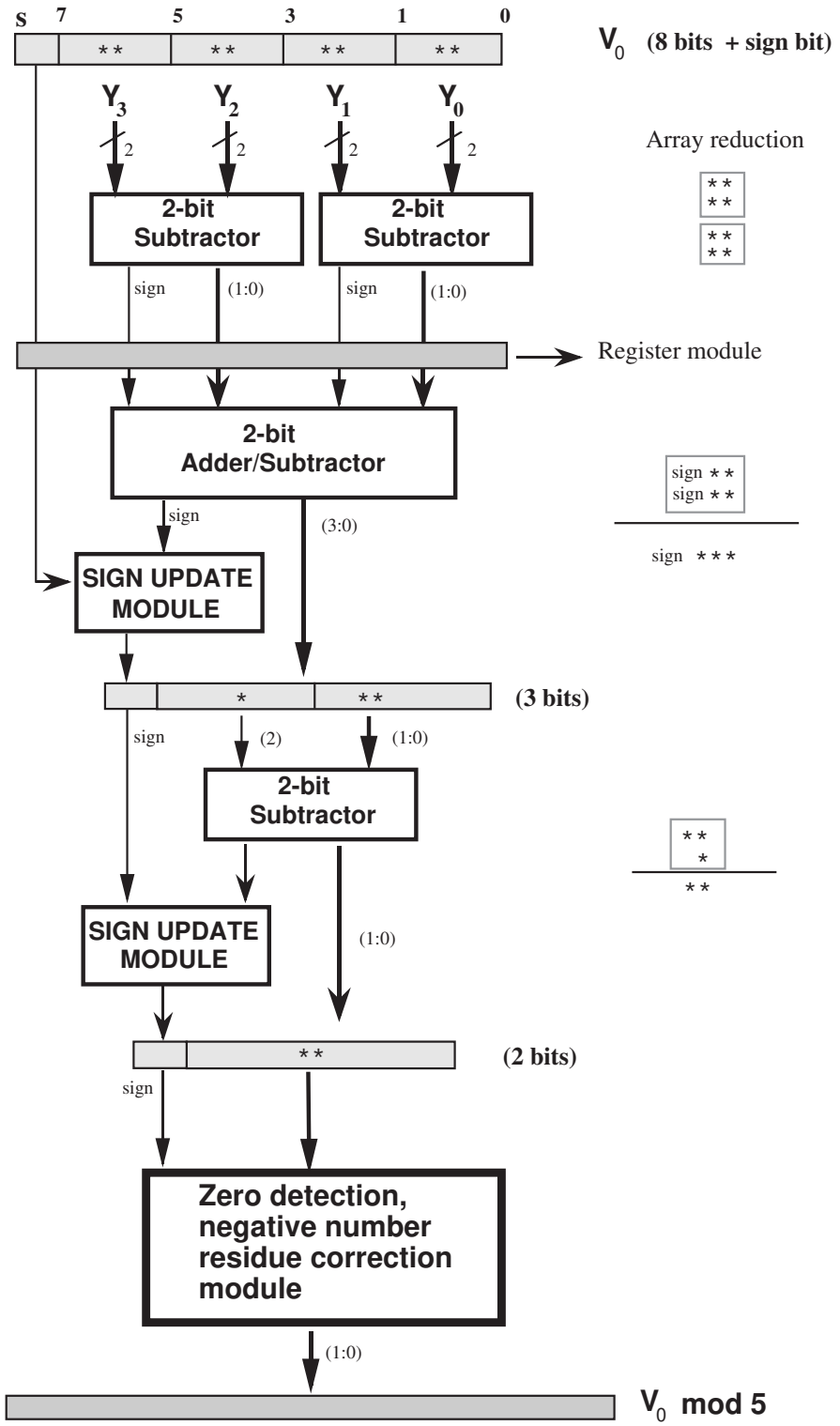

Figure 4 *Schematic diagram of* V_0 mod 5 *unit where* V_0 *is an 8-bit number. ('*' denotes the individual bits of the operand).*

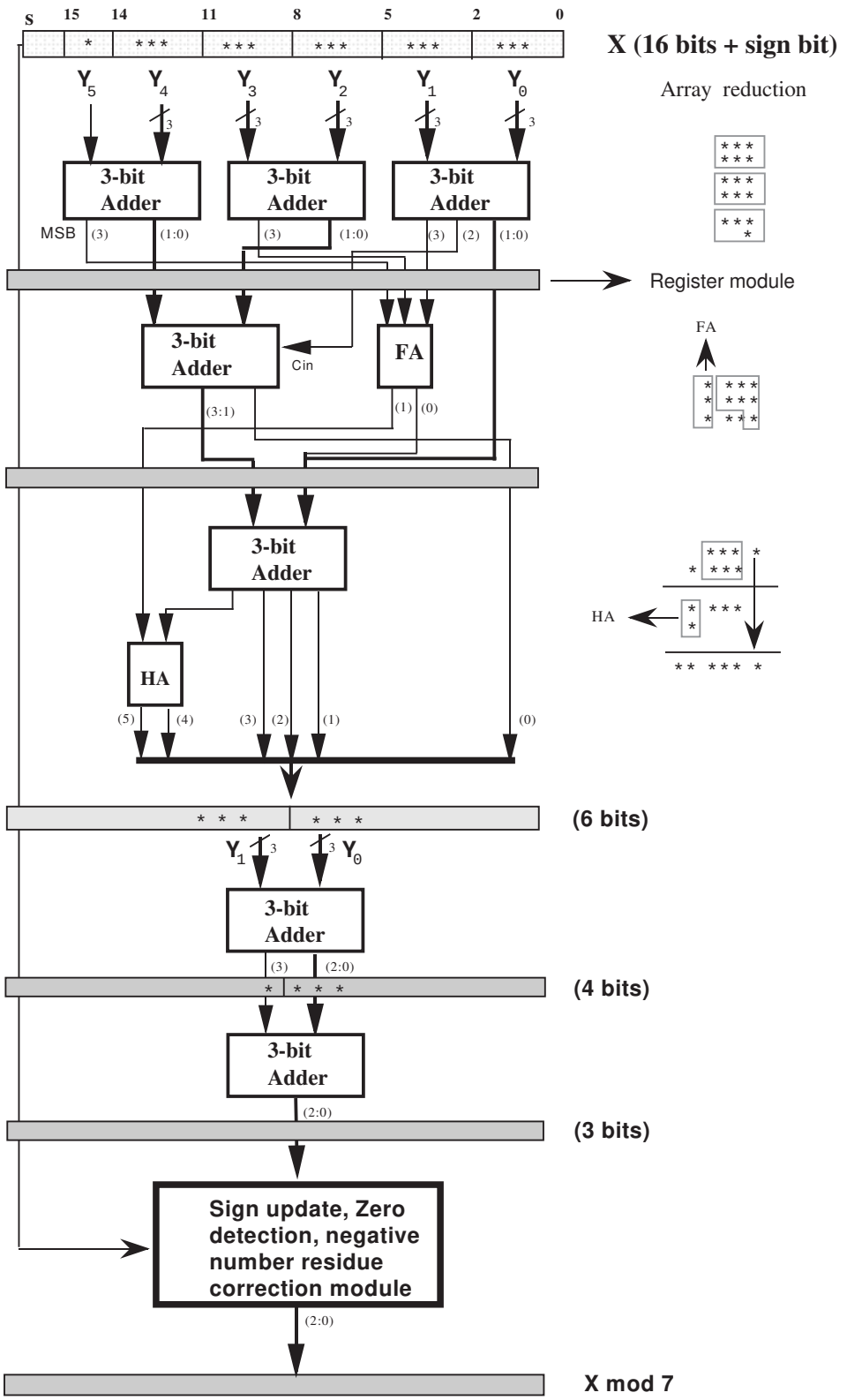

Figure 5 *Schematic diagram of* X mod 7 *unit where* X *is a 16-bit signed integer.*

Table 2 *Statistics of the implemented* 16*-bit residue extractor*

Technology	3μ CMOS3DLM
Area of the chip	4545.1x4545.1 $\mu^2 m$
No. of pins	38
No. of transistors	5156
Measured Propagation delay	108.6ns
Dynamic Power dissipation/Mhz	30mwatts

Table 3 *AC characteristics of the residue extractor*

Output Group	Max. Propagation Delay
MOD10	108.6ns
MOD9	108.3ns
MOD7	107.9ns
MOD6	75.4ns
MOD5	104.7ns
MOD3	62.3ns

6.4. VLSI Chip

The micrograph of the implemented 16-bit residue extractor is shown in Fig. 6 and some related technical details are summarized in Table 2. The performance of the chip was characterized by clocking input data at a period of 200ns (non-pipelined case). A timing analysis was conducted on the chip using an automatic test equipment. The measured worst case delay of the sub-units of the chip are detailed in Table 3. With a maximum delay of approximately 109ns, the chip gives a throughput in excess of 9 million operations per second (MOPS).

7. Discussion

Notwithstanding the tremendous progress in contemporary VLSI memory technology, ROM based techniques [6] are expensive and infeasible for large operands since memory requirements increase exponentially with operand length. In this work, we have proposed two new VLSI architectures for implementing X mod m for $m = \beta^k - 1$ or $\beta^k + 1$ and presented a practical and scalable hardware circuit for for the binary number system. We have also shown that the range of m can be extended using the CRT. In most practical cases, the representative length of the operand X is larger than that of m. Hence, the ROM's cost for implementing the RMF will be less expensive

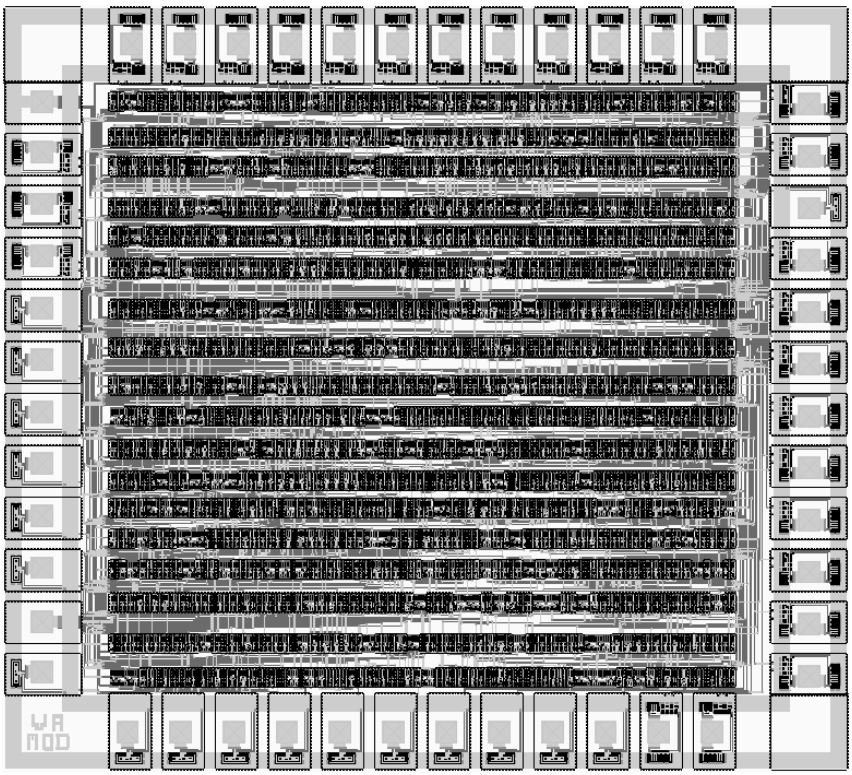

Figure 6 *Micrograph of a* 16*-bit residue extractor implemented in 3* μm *CMOS3DLM technology*

Table 4 *Synthesis method of* $X \bmod m$ *for typical values of* $m \leq 33$.

m	Type		m	Type	
2	2^k	$(k = 1)$	18	$(\langle X \rangle_2, \langle X \rangle_9)$	CRT(2,9)
3	$2^k - 1$	$(k = 2)$	19	–	
4	2^k	$(k = 2)$	20	$(\langle X \rangle_4, \langle X \rangle_5)$	CRT(4,5)
5	$2^k + 1$	$(k = 2)$	21	$(\langle X \rangle_3, \langle X \rangle_7)$	CRT(3,7)
6	$(\langle X \rangle_2, \langle X \rangle_3)$	CRT(2,3)	22	$(\langle X \rangle_2, \langle X \rangle_{11})$	CRT(2,11)
7	$2^k - 1$	$(k = 3)$	23	–	
8	2^k	$(k = 3)$	24	$(\langle X \rangle_3, \langle X \rangle_8)$	CRT(3,8)
9	$2^k + 1$	$(k = 3)$	25	–	
10	$(\langle X \rangle_2, \langle X \rangle_5)$	CRT(2,5)	26	–	
11	–		27	–	
12	$(\langle X \rangle_3, \langle X \rangle_4)$	CRT(3,4)	28	$(\langle X \rangle_4, \langle X \rangle_7)$	
13	–		29	–	
14	$(\langle X \rangle_2, \langle X \rangle_7)$	CRT(2,7)	30	$(\langle X \rangle_5, \langle X \rangle_6)$	CRT(5,6)
15	$2^k - 1$ or	$(k = 4)$	31	$2^k - 1$	$(k = 5)$
	$(\langle X \rangle_3, \langle X \rangle_5)$	CRT(3,5)			
16	2^k	$(k = 4)$	32	$2^k - 1$	$(k = 5)$
17	$2^k + 1$	$(k = 4)$	33	$2^k + 1$	$(k = 5)$

compared to a complete look-up. In Table 4, we summarize a typical moduli set that can be implemented using the schemes discussed in this paper for small values of m. The prime factors of m which are synthesizable through the CRT are given in brackets for clarity.

The proposed structures, in addition to their regularity and modularity, have significant advantages in terms of speed and area over the model proposed in [7]. It has been reported that the VLSI circuit designed by Alia *et al.* [7] for integer modulo m operation has a response time of less than 200 ns for 32-bit numbers. The area and time complexity of their architecture is given as $\mathcal{O}(n^2/T_M^2)$ and $\mathcal{O}(T_M)$ respectively when T_M is in the range $[\log n, \sqrt{n}]$. It has been shown in this paper that the time complexity for the two methods is in the range $[\mathcal{O}(\log n), \mathcal{O}(\log n \log\log n)]$ while the area complexity is in the range $[\mathcal{O}(n), \mathcal{O}(n \log n)]$. The measured propagation delay in our circuit for 16-bit input operands is less than 109ns in 3μ CMOS technology. Our design uses high speed carry look ahead adders [20] and does not involve any multipliers as advocated in [7]. Furthermore, the proposed structures can handle signed integers and are amenable for pipeline implementation. Hence, wider operand range and higher throughput can be realized. The demerits of our model are its restricted range for m and the inability to compute residues of prime numbers which are not of the form $2^k, 2^k \pm 1$ in contrast to the generalized approach of [7].

8. Conclusion

The technological advantages of VLSI has made RNS implementation a viable proposition in terms of speed, cost, power dissipation and chip density. The basic contribution of this work is the design approach for calculating $X \bmod m$ when m is of the form $\beta^k, \beta^k \pm 1$. Two algorithms, namely the recursive and partition methods have been proposed and expressions for the asymptotic VLSI complexity for their implementation have been derived. The methodology discussed in this paper has been practically applied in the design of $X \bmod m$ arithmetic units for a port selector module in a $1.2\mu m$ CMOS Hypercycle routing chip [21] where X is a 16-bit operand with $m = 2, 3, 4$. The cases $m = 2, 4$ are straightforward while the $X \bmod 3$ unit has been implemented using the recursive approach for 4-bit partitions with a two level residue adder tree. The throughput of the residue extractor module in the above chip exceeds 50 MOPS as evidenced by timing simulation results. Technological scaling and pipeline implementation will provide a many-fold increase in the performance of the residue extractor. Additionally, the algorithms proposed here can be extended to non-binary systems when multi-level logic circuits become viable for implementation. Other interesting applications of this work include design of circuits for constant division algorithms [22].

Acknowledgement

The authors wish to thank the Canadian Microelectronics Corporation for its assistance in fabricating the VLSI chips discussed in this paper. They also acknowledge Mr. K. Jones, Mr. W. A. Keddy, Mr. R. Kelly, Mr. T. Gore and Mrs. A. Neville for their technical help in the VLSI/Microelectronics laboratory. The authors thank the reviewers for their useful suggestions which have greatly improved the clarity and presentation of this work. The financial support of the Natural Sciences and Engineering Research Council of Canada and the Institute for Robotics & Intelligent Systems of the National Network of Centres of Excellence is gratefully acknowledged.

References

[1] F. J. Taylor, "Residue Arithmetic: A tutorial with examples", *IEEE Comp.*, May 1984, **17** (5), pp. 50-62.

[2] A. V. Curiger, H. Bonnenberg, and H. Kaeslin, "Regular VLSI Architectures for multiplication Modulo $(2^n + 1)$", *IEEE J. Solid-State Circuits*, 1991, **26** (7), pp. 990-994.

[3] W. K. Jenkins, and B. J. Leon, " The use of residue number systems in the design of finite impulse response digital filters", *IEEE Trans. Circuits Sys.*, 1977, **CAS-24**, pp. 191-201.

[4] M. A. Soderstrand, "A high speed low-cost recursive digital filtering using Residue Arithmetic", *Proc. IEEE*, July 1977, **65**, pp. 1065-1067.

[5] C. H. Huang, D. G. Patterson, H. E. Rauch, J. W. Teague, and D. F. Fraser, " Implementation of a fast digital processor using the residue number system", *IEEE Trans. Circuits Sys.*, 1981, **CAS-28**, pp. 32-38.

[6] M. A. Bayoumi, G. A. Jullien, and W. C. Miller, "A VLSI model for Residue Number System Architectures", *Integration*, 1984, **2**, pp. 191-211.

[7] G. Alia, and E. Martinelli, "A VLSI Structure for X(mod m) Operation", *Journal of VLSI Signal Processing*, 1990, **1**, pp. 257-264.

[8] F. J. Taylor, "A VLSI residue arithmetic multiplier", *IEEE Trans. Comp.*, 1982, **C-31**, pp. 540-546.

[9] N. S. Szabo, and R. I. Tanaka, *Residue Arithmetic and its Applications to Computer technology*, McGraw Hill, New York, 1967.

[10] J. H. Mcclellan, and C. M. Rader, *Number Theory in Digital signal processing*, Prentice Hall, New York, 1979.

[11] S. Waser, and M. Flynn, *Introduction to Arithmetic for Digital System designers*, Holt, Rinehart and Winston, Chicago, 1982, pp. 55

[12] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation to Computer Science*, Addison-Wesley, Reading, Massachusetts, 1989, pp. 126-133.

[13] D. E. Knuth, *The Art of Computer Programming : Seminumerical Algorithms*, vol. 2, Addison-Wesley, Reading-Mass 1981, pp. 248-256.

[14] J. Wakerly, "Error Detecting Codes, Self-Checking Circuits and Applications" in *Computer Design and Architecture Series*, (North-Holland, 1978), pp. 71-74

[15] J. D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, Maryland, 1984.

[16] S. Dormido, and M. A. Canto, "Synthesis of generalized parallel counters", *IEEE Trans. Comp.*, Sept 1981, **C-30**, pp. 699-703.

[17] K. Efe, "Multi-Operand Addition with conditional sum logic" in *Proc.* 5^{th} *Symposium Comp. Arith.*, Ann Arbor, May 1981, pp. 251-255.

[18] L. Dadda, "Multiple Addition of Binary Serial numbers", *Proc.* 4^{th} *Ann. Symp. Comp. Arith.*, Long Beach, 1978, pp. 140-148.

[19] M. Dugale, "VLSI implementation of Residue Adders based on binary adders", *IEEE Trans. Circ. and Sys.*, May 1992, vol. 2, **39**, pp. 325-329

[20] R. P. Brent, and H. T. Kung, "A Regular Layout for Parallel Adders", *IEEE Trans. Comp.*, March 1982, **C-31** (3), pp. 260-264.

[21] N. J. Dimopoulos, D. Radvan, and R. Sivakumar, "Implementation of Routing Engine for Hypercycle based Interconnection Networks", *Proc. 1991 Canadian Conf. on VLSI*, Kingston, Aug. 1991, pp. 6.4.1-6.4.7.

[22] P. Srinivasan, and F. E. Petry, "Constant Division Algorithms", *IEE Proc. Comp. and Digital Tech.*, Nov 1994, vol. 141 (6), pp. 334-340.