Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors, pp. 18–21, IEEE Computer Society Press, Cambridge, Massachusetts, October 2–4, 1989.

A Fast Algorithm for Mixed-Radix Conversion in Residue Arithmetic

Çetin K. Koç
Department of Electrical Engineering
University of Houston
Houston, TX 77204
cetin@izmir.ee.uh.edu

Abstract

An algorithm based on a partitioning of the coefficient matrix when the mixed-radix conversion problem is cast as a set of linear congruent equations is presented. The presented algorithm partitions the moduli set into disjoint subsets such that the product of the moduli in each subset is less than the largest integer representable by the computer. We show that, with this partitioning strategy, mixed-radix representation of a residue number can be computed using less than $O(n^2)$ arithmetic steps where n is the cardinality of the moduli set. We also show that if a 'good' partitioning exists then the algorithm requires only $O(n^{1.5})$ arithmetic steps. The presented algorithm is particularly suitable for single processor implementation of algorithms from the residue number systems applications.

1 Introduction

Since addition, subtraction, and multiplication can be performed without carry propagation, residue number systems (RNS) are preferred to weighted number systems for doing arithmetic on large integers, and for implementing digital signal processing algorithms [11, 13]. Residue arithmetic is also used to find solutions of equations over an integral domain [3, 2, 8, 9]. This allows computation of the exact solution of linear equations when the matrix of the coefficients is ill-conditioned.

A drawback of RNS, however, is that a number represented in residue notation does not have magnitude information, and thus should be converted to a weighted number system to extract this information. The conversion techniques from a residue notation to a weighted notation, and vice versa, are needed in almost all applications employing residue arithmetic. For example, in hardware implementations of signal processing algorithms, the input and output signals are usually in analog form; thus the numbers have to be represented in a weighted notation before converting to analog. For software implementation of RNS algorithms, it is required that the residue numbers be converted to decimal.

The conversion from a weighted representation to a residue representation is achieved by concurrent application of integer division operations, a very simple and efficient process. The conversion from a residue notation to a weighted notation is, however, not that simple. The methods for this conversion are based on two different constructive proofs of the *Chinese Remainder Theorem* (CRT). In the first case, the number is converted to a *single-radix* weighted number system; in the second case it is converted to a *mixed-radix* weighted number system [8]. The similarity between interpolation of polynomials and Chinese remaindering of integers is well-known [10, 2, 8, 1]. While the Lagrange polynomial interpolation formula is the analogue of the single-radix conversion algorithm for integer Chinese remaindering, the Aitken algorithm for the Newton polynomial interpolation is the analogue of the mixed-radix conversion algorithm. The mixed-radix conversion algorithm is attributed to Garner [6] (see [8, 9]). Henceforth, we refer to this algorithm as the *Garner Algorithm*.

We define W as the largest integer which can be operated on by the arithmetic unit of the computer. When each of the moduli is less than W, Garner's algorithm requires $O(n^2)$ arithmetic steps [10, 9]. However, with the emergence of 32–bit and 64–bit microprocessors and numeric coprocessors which are capable of operating on integers as large as 2^{64}, it is more likely that W is significantly larger than each of the moduli; therefore Garner's algorithm does not fully utilize the arithmetic unit of the computer.

Here we present an algorithm which partitions the moduli set into disjoint subsets such that the product of the moduli in each subset is less than W. We show that if there is a nontrivial partitioning (i.e., the number of subsets is strictly less than n), the presented algorithm always requires less than $O(n^2)$ arithmetic operations, and that if a 'good' partitioning exists, the algorithm requires only $O(n^{1.5})$ arithmetic steps.

This paper is organized as follows: in §2, we give the Garner algorithm and show that it requires $O(n^2)$ arithmetic operations. In §3 we present an algorithm which partitions a large mixed-radix conversion problem into mixed-radix problems of smaller size. We prove that, under favorable conditions, this partitioning approach yields an algorithm which requires $O(n^{1.5})$ arithmetic operations. Finally in §4, we discuss the issues regarding a practical implementation of this algorithm, and point out some directions for future research.

2 The Garner Algorithm

We are given:

- The moduli set $\{m_1, m_2, \ldots, m_n\}$ consisting of n pairwise relatively prime numbers, and
- The residues of a weighted number u with respect to each modulus

$$u_i \equiv u \pmod{m_i} \text{ for } 1 \le i \le n \ .$$

The number u then can be computed using Garner's algorithm as follows:

Step 1. Compute constants c_{ij} for $1 \le i < j \le n$ where

$$c_{ij} m_i \equiv 1 \pmod{m_j} \ .$$

Step 2. Compute

$$
\begin{aligned}
v_1 &\equiv u_1 \pmod{m_1} \\
v_2 &\equiv (u_2 - v_1)c_{12} \pmod{m_2} \\
v_3 &\equiv ((u_3 - v_1)c_{13} - v_2)c_{23} \pmod{m_3} \\
&\vdots \\
v_n &\equiv (\cdots((u_n - v_1)c_{1n} - v_2)c_{2n} - \cdots - v_{n-1})c_{n-1,n} \pmod{m_n}
\end{aligned}
$$

Thus, number u given in residue representation $(u_1, u_2, \ldots, u_n)$ now can be represented in mixed-radix notation as $(v_1, v_2, \ldots, v_n)$. This representation of u has magnitude information since

$$u = v_1 + v_2 m_1 + v_3 m_1 m_2 + \cdots + v_n m_1 m_2 \cdots m_{n-1} \ . \qquad (1)$$

Furthermore, the single-radix representation of this mixed-radix number can be found by applying Horner's algorithm to the above formula. The constants c_{ij} are the multiplicative inverses of m_i modulo m_j for all $1 \leq i < j \leq n$ and can be computed using Euclid's algorithm (see [8, 9]). We denote the operation to find the multiplicative inverse of a modulo b by INVERSE(a, b).

For the timing analysis of the algorithms on a single processor machine, we assume the following:

- an arithmetic operation ($\in \{+, -, \div, \times\}$) on numbers $\leq W$ is assumed take 1 unit time, defined as 1 *arithmetic step*,
- for operations on numbers $> W$, multi-precision arithmetic is implemented.

Thus, we see that according to this model:

1. If $a, b \leq W$, then Euclid's algorithm requires $O(\log W)$ arithmetic operations to compute INVERSE(a, b). Since W is independent of n we assume that INVERSE operation on single-precision numbers takes only $O(1)$ arithmetic operations.
2. If $A, B \leq W$, then A mod B can be computed in $O(1)$ time since it can be achieved by a division operation.
3. If $v_i, m_i \leq W$ for $1 \leq i \leq n$, then the application of Horner's algorithm to compute single-radix representation of u requires $O(n^2)$ arithmetic steps using multi-precision arithmetic [9].

We define V_{ij} for $0 \leq i < j \leq n$ such that $V_{0j} = u_j$ for $1 \leq j \leq n$ and $V_{i-1,i} = v_i$ for $1 \leq i \leq n$. V_{ij} for $1 \leq i < j \leq n$ are the temporary values of v_j resulting from the operations in Step 2. This way, we build a triangular table of values with diagonal entries $v_i = V_{i-1,i}$ for $2 \leq i \leq n$. This table is similar to the divided difference table in computing the coefficients of the Newton interpolating polynomial, i.e., the divided differences. The entries of this table are named *multiplied differences* [10]. For $n = 4$, it can be given as follows:

$$V_{12} \equiv (V_{02} - V_{01})c_{12} \pmod{m_2}$$

$$V_{13} \equiv (V_{03} - V_{01})c_{13} \ , \ V_{23} \equiv (V_{13} - V_{12})c_{23} \pmod{m_3}$$

$$V_{14} \equiv (V_{04} - V_{01})c_{14} \ , \ V_{24} \equiv (V_{14} - V_{12})c_{24} \ , \ V_{34} \equiv (V_{24} - V_{23})c_{34} \pmod{m_4}$$

We give Garner's algorithm in a Pascal-like language below.

```
PROCEDURE GARNER(u_i, m_i ; 1 ≤ i ≤ n)
FOR j = 1 TO n DO
    BEGIN
    V_0j = u_j
    END FOR
FOR i = 1 TO n − 1 DO
    BEGIN
    FOR j = i + 1 TO n DO
        BEGIN
        c_ij ≡ INVERSE(m_i, m_j)
        V_ij ≡ (V_i−1,j − V_i−1,i)c_ij   (mod m_j)
        END FOR
    END FOR
FOR i = 1 TO n DO
    BEGIN
    v_i = V_i−1,i
    END FOR
END PROCEDURE
```

Theorem 1 *Given the moduli $m_1, m_2, \ldots, m_n$ and the remainders $u_1, u_2, \ldots, u_n$ such that $m_i \leq W$ for $1 \leq i \leq n$, the mixed-radix number representation $(v_1, v_2, \ldots, v_n)$ of u, i.e., the multiplied differences of u, can be computed in $O(n^2)$ arithmetic steps with Garner's algorithm.*

Proof Garner's algorithm computes the terms V_{ij} for $1 \leq i < j \leq n$ by performing the following operations on single-precision operands:

$$c_{ij} \equiv \text{INVERSE}(m_i, m_j) \ , \qquad (2)$$

$$V_{ij} \equiv (V_{i-1,j} - V_{i-1,i})c_{ij} \pmod{m_j} \ . \qquad (3)$$

There are exactly $\frac{n(n-1)}{2}$ entries in the table. Each entry is computed using $O(1)$ arithmetic operations: INVERSE, subtraction, and multiplication on single-precision numbers. Thus, the total number of arithmetic operations required for the computation of multiplied differences by Garner's algorithm is $O(n^2)$. □

We note that the above proof holds for *preconditioned* Chinese remaindering as well. In this case the multiplicative inverses c_{ij} are precomputed. The arithmetic operations executed now are given by (3).

3 A Partitioning Algorithm

In this section, we formulate the mixed-radix conversion problem as a system of linear congruence equations and show that it can be partitioned into smaller size mixed-radix conversion problems and some additional arithmetic operations.

Recall equation (1)

$$u = v_1 + v_2 m_1 + v_3 m_1 m_2 + \cdots + v_n m_1 m_2 \cdots m_{n-1} \ .$$

Coefficients v_i for $1 \leq i \leq n$ can be obtained by solving

$$
\begin{aligned}
v_1 &\equiv u_1 \pmod{m_1} \\
v_1 + v_2 m_1 &\equiv u_2 \pmod{m_2} \\
v_1 + v_2 m_1 + v_3 m_1 m_2 &\equiv u_3 \pmod{m_3} \\
&\vdots \\
v_1 + v_2 m_1 + \cdots + v_n m_1 m_2 \cdots m_{n-1} &\equiv u_n \pmod{m_n}
\end{aligned}
$$

We represent the above linear system of equations in matrix notation as

$$\mathbf{M}\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_n \end{bmatrix} \equiv \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{bmatrix} \pmod{\begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ \vdots \\ m_n \end{bmatrix}}$$

where

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & m_1 & 0 & 0 & \cdots & 0 \\ 1 & m_1 & m_1m_2 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & m_1 & m_1m_2 & m_1m_2m_3 & \cdots & m_1m_2m_3\cdots m_{n-1} \end{bmatrix}$$

or more compactly, $\mathbf{Mv} \equiv \mathbf{u} \pmod{\mathbf{m}}$. This system is solved by applying Garner's algorithm to the integers $u_1, u_2, \ldots, u_n$ using the moduli set $m_1, m_2, \ldots, m_n$.

We assume here that $n = k_1 + k_2 + \cdots + k_q$ and define the quantities $l_i = k_1 + k_2 + \cdots + k_i$ for $1 \le i \le q$ and $l_0 = 0$. We partition the moduli set into q disjoint subsets

$$\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_q\}$$

such that the cardinality of each $\mathcal{M}_i$ is equal to k_i for $1 \le i \le q$ and the product of the moduli in each subset $\mathcal{M}_i$ is less than W for $1 \le i \le q$, i.e.,

$$\mathcal{M}_i = \{m_{l_{i-1}+1}, m_{l_{i-1}+2}, \ldots, m_{l_i}\}$$

with

$$m_{l_{i-1}+1} \times m_{l_{i-1}+2} \times \ldots \times m_{l_i} \le W$$

for $1 \le i \le q$. This way We partition matrix M into $k_i \times k_j$ dimension matrices $\mathbf{M}_{ij}$ for $1 \le j \le i \le q$, and vectors v, u, and m into k_i dimension vectors $\mathbf{v}_i, \mathbf{u}_i, \mathbf{m}_i$ for $1 \le i \le q$. Thus,

$$\begin{bmatrix} \mathbf{M}_{11} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{M}_{21} & \mathbf{M}_{22} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{M}_{31} & \mathbf{M}_{32} & \mathbf{M}_{33} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{M}_{q1} & \mathbf{M}_{q2} & \mathbf{M}_{q3} & \cdots & \mathbf{M}_{qq} \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ \vdots \\ \mathbf{v}_q \end{bmatrix} \equiv \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ \vdots \\ \mathbf{u}_q \end{bmatrix} \bmod \begin{bmatrix} \mathbf{m}_1 \\ \mathbf{m}_2 \\ \mathbf{m}_3 \\ \vdots \\ \mathbf{m}_q \end{bmatrix}$$

where matrices $\mathbf{M}_{ii}$ are lower-triangular and the zero matrices are of the same dimension as the $\mathbf{M}_{ij}$ matrices. This partitioning of the problem yields an algorithm where we first solve the system $\mathbf{M}_{11}\mathbf{v}_1 \equiv \mathbf{u}_1 \pmod{\mathbf{m}_1}$ using PROCEDURE GARNER and then compute

$$v_1 + v_2m_1 + \cdots + v_{k_1}m_1m_2\cdots m_{k_1-1} \tag{4}$$

using Horner's algorithm. Since all $v_i, m_i \le W$ for $1 \le i \le k_1$, and the maximum value (4) can attain is

$$(m_1 - 1) + m_1(m_2 - 1) + \cdots + m_1m_2\cdots m_{k_1-1}(m_{k_1} - 1) =$$

$$= m_1m_2m_3\cdots m_{k_1} - 1 \le W \ , \tag{5}$$

the application of Horner's algorithm to compute (4) requires only single-precision arithmetic. We then update the values of $\mathbf{u}_i$ for $2 \le i \le q$ according to the formulae (2) and (3) in a block fashion. The computation of the second block-column is similar to the first one. We give the partitioning algorithm below.

PROCEDURE PARTITIONED GARNER $(u_i, m_i \ ; \ 1 \le i \le n)$
FOR $i = 1$ TO q DO
  BEGIN
1: $(v_j \ ; \ l_{i-1} + 1 \le j \le l_i) =$ GARNER$(u_j, m_j \ ; \ l_{i-1} + 1 \le j \le l_i)$
2: $s_i = v_{l_{i-1}+1} + v_{l_{i-1}+2} \times m_{l_{i-1}+1} + \cdots$
$\cdots + v_{l_i} \times m_{l_{i-1}+1} \times m_{l_{i-1}+2} \times \cdots \times m_{l_i-1}$
  FOR $j = l_{i-1} + 1$ TO n DO
    BEGIN
3: $u_j \equiv u_j - s_i \pmod{m_j}$
    END FOR
4: $t_i = m_{l_{i-1}+1} \times m_{l_{i-1}+2} \times \cdots \times m_{l_i}$
  FOR $j = l_{i-1} + 1$ TO n DO
    BEGIN
5.1: $t_{ij} \equiv$ INVERSE(t_i, m_j)
5.2: $u_j \equiv t_{ij}u_j \pmod{m_j}$
    END FOR
  END FOR
END PROCEDURE

Theorem 2 *If there is a partitioning of the moduli set $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_q\}$ such that the cardinality of $\mathcal{M}_i$ is k_i and the product of the moduli in each $\mathcal{M}_i$ is $\le W$ for $1 \le i \le q$ then the partitioned Garner algorithm requires*

$$O(k_1^2 + k_2^2 + \cdots + k_q^2 + nq) \tag{6}$$

arithmetic steps to compute the multiplied differences v_i for $1 \le i \le n$.

Proof In step 1, the call to PROCEDURE GARNER occurs q times for $i = 1, 2, \ldots, q$ where at instance i a problem of size k_i is solved. The variables involved in the computation of $\mathbf{v}_i$ are all single-precision numbers. It follows from Theorem 1 that this step requires $O(k_i^2)$ arithmetic step for a particular value of i.

In step 2, we find s_i by applying Horner's algorithm to the multiplied differences computed in step 1 and the moduli from set $\mathcal{M}_i$. According to equation (5) the maximum value s_i can attain is less than W, making s_i is a single-precision number. Since all input variables and the temporary values resulting from the application of Horner's algorithm are in single-precision, this step requires only $O(k_i)$ arithmetic operations.

In step 3, the j loop executed $n - l_i$ times for a particular value of i, where at each step a single-precision subtraction (followed by a correction to bring the result back to the desired range) is performed. Thus in step 3, $O(1)(n - l_i)$ arithmetic operations are performed. Since we have $n - l_i = k_{i+1} + k_{i+2} + \cdots + k_q$, the number of arithmetic operations becomes

$$O(1) \sum_{j=i+1}^{q} k_i \ . \tag{7}$$

In step 4, the product of all moduli in the subset $\mathcal{M}_i$ can be found by performing $k_i - 1$ single-precision multiplications.

In step 5 , similar to step 3, we execute the FOR loop for $n - l_i$ times where at each step $O(1)$ operations are performed, namely an INVERSE operation and a multiplication involving single-precision integers. Thus the number of arithmetic steps required by Step 5 is also given by (7).

For $1 \le i \le n$, we sum the number of arithmetic operations at each step and find the total number of arithmetic operations required by the partitioned Garner algorithm as

$$T = \sum_{i=1}^{q}(O(k_i^2) + O(k_i) + O(1)\sum_{j=i+1}^{q} k_i)$$
$$= O(\sum_{i=1}^{q} k_i^2 + \sum_{i=1}^{q} k_i + \sum_{i=1}^{q}\sum_{j=i+1}^{q} k_i)$$
$$= O(\sum_{i=1}^{q} k_i^2 + q\sum_{i=1}^{q} k_i) = O(k_1^2 + k_2^2 + \cdots + k_q^2 + nq) .$$

□

We now give another theorem regarding the minimum value of expression (6).

Theorem 3 *The number of arithmetic operations required by the partitioned Garner algorithm is $O(n^{1.5})$ when $k_i = \frac{n}{q}$ for $1 \leq i \leq q$ and $q = \sqrt{n}$. Furthermore, this is the minimum value of (6).*

Proof Expression $k_1^2 + k_2^2 + \cdots + k_q^2$ takes its minimum value when $k_i = k$ for $1 \leq i \leq q$. Since we have $k_1 + k_2 + \cdots + k_q = n$ it follows that $k = \frac{n}{q}$. The number of arithmetic operations required by PARTITIONED GARNER procedure then becomes

$$T(q) = O((\frac{n}{q})^2 q + nq) = O(\frac{n^2}{q} + nq) . \quad (8)$$

The minimum of (8) can be found by solving the equation

$$T'(q) = -\frac{n^2}{q^2} + n = 0 ,$$

which gives the optimum value of $q^* = \sqrt{n}$. Thus, we find $T_{min} = O(\frac{n^2}{\sqrt{n}} + n\sqrt{n}) = O(n^{1.5})$. □

The single-radix representation of number u can also be computed if the temporary values s_i, t_i for $1 \leq i \leq q$ in PROCEDURE PARTITIONED GARNER are saved. Since we have

$$u = s_1 + s_2 t_1 + s_3 t_1 t_2 + \cdots + s_q t_1 t_2 \cdots t_{q-1} ,$$

we can apply Horner's algorithm to compute u. The temporary and the resulting values for this computation exceed W; thus we need to implement multiple-precision arithmetic. The application of Horner's algorithm using multiple-precision arithmetic requires $O(q^2)$ arithmetic steps to compute the single-radix representation of u [9]. If $q = \sqrt{n}$, then this computation takes only $O(q^2) = O(n)$ arithmetic steps. When the partitioned Garner algorithm is used to compute the single-radix representation of u, the number of arithmetic operations required is dominated by $O(n^{1.5})$ under the assumptions of Theorem 3.

4 Conclusions

We have proved that if a partitioning of the moduli set $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_q\}$ exists such that

1. cardinality of each $\mathcal{M}_i$ is $k_i = k$, and
2. $k = q = O(\sqrt{n})$, and
3. the product of the moduli in each subset $\mathcal{M}_i$ is less than W, the largest number representable by our computer,

then the partitioned Garner algorithm computes the multiplied differences using $O(n^{1.5})$ arithmetic operations. Furthermore, the single-radix representation of number u can also be computed by the application of Horner's algorithm in multiple-precision arithmetic using another $O(n)$ arithmetic steps. When the moduli set cannot be partitioned because the product of the moduli in any subset of $\mathcal{M}$ is larger than W, the number of arithmetic operations required by the partitioned Garner algorithm becomes $O(n^2)$. Thus, a practical implementation will have the running time as $O(n^\alpha)$ where $1.5 \leq \alpha \leq 2$.

The partitioning of the moduli set may be complicated since it is related to the *set partitioning problem* which is NP-complete (see Problem SP12 in [5], pp. 223). We are currently investigating this issue. We are also implementing the Garner and the partitioned Garner algorithm on a scientific workstation. The results of this work and some strategies for the partitioning process will be reported in the near future.

We note that several implementations of Garner's algorithm are given in the literature. These implementations are mostly hardware oriented, and use table lookup techniques and thus put *a priori* restrictions on the size and cardinality of the moduli [12, 7, 4]. The partitioning algorithm described in this paper does not have such restrictions; other than the fact that the running time of the algorithm may exceed $O(n^{1.5})$.

References

[1] A. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[2] E. H. Bareiss, "Computational Solutions of Matrix Problems Over an Integral Domain," *J. Inst. Maths. Applics.*, No. 10, pp. 68–104, 1972.

[3] I. Borosh and A. S. Fraenkel, "Exact Solutions of Linear Equations with Rational Coefficients by Congruence Techniques," *Mathematics of Computation*, Vol. 20, No. 93, pp. 107–112, January 1966.

[4] N. B. Chakraborti, J. S. Soundararajan, and A. L. N. Reddy, "An Implementation of Mixed-Radix Conversion for Residue Number Applications," *IEEE Trans. on Computers*, Vol. C-35, No. 8, pp. 762–764, Augusts 1986.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., 1979.

[6] H. L. Garner, "The Residue Number System," *IRE Trans. Electronic Computers*, Vol. EL-8, No. 6, pp. 140–147, June 1959.

[7] C. H. Huang, "A Fully Parallel Mixed-Radix Conversion Algorithm for Residue Number Applications," *IEEE Trans. on Computers*, Vol. C-32, No. 4, pp. 398–402, April 1983.

[8] D. E. Knuth, *The Art of Computer Programming*, Volume 2, *Seminumerical Algorithms*, 2nd Edition, Addison-Wesley Publishing Company, 1981.

[9] J. D. Lipson, *Elements of Algebra and Algebraic Computing*, Addison-Wesley Publishing Company, 1981.

[10] J. D. Lipson, "Chinese Remainder and Interpolation Algorithms," *Proc. 2nd Symp. Symbolic Algebraic Manipulation*, pp. 372–391, 1971.

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

[12] F. J. Taylor, "An Efficient Residue-To-Decimal Converter," *IEEE Trans. on Circuits and Systems*, Vol. CAS-28, No. 12, pp. 1164–1169, December 1981.

[13] F. J. Taylor, "Residue Arithmetic: A Tutorial with Examples," *IEEE Computer Mag.*, Vol. 17, pp. 50-62, May 1984.