Design of Residue Generators and Multioperand Modular Adders Using Carry-Save Adders

Stanisław J. Piestrak

Abstract—**Residue generator is an essential building block of encoding/decoding circuitry for arithmetic error detecting codes and binary-to-residue number system (RNS) converter. In either case, a residue generator is an overhead for a system and as such it should be built with minimum amount of hardware and should not compromise the speed of a system. Multioperand modular adder (MOMA) is a computational element used to implement various operations in digital signal processing systems using RNS.**

In this paper, a comprehensive study of new residue generators and MOMA's is presented. The design methods given here take advantage of the periodicity of the series of powers of 2 taken modulo A (A is a module). Four design schemes of the n-input residue generators mod A, which are best suited for various pairs of n and A, are proposed. Their pipelined versions can be clocked with the cycle determined by the delay of a full-adder and a latch. A family of design methods for parallel and word-serial, using similar concepts, is also given. Both classes of circuits employ new highly-parallel schemes using carry-save adders with end-around carry and a minimal amount of ROM and are well-suited for VLSI implementation. They are faster and use less hardware than similar circuits known to date. One of the MOMA's can be used to build a high-speed residue-to-binary converter based on the Chinese remainder theorem.

Index Terms—**Addition modulo A, arithmetic codes, binary-to-residue number system converter, carry-save adders, Chinese remainder theorem, generator modulo A, modular arithmetic, multioperand modular addition, residue arithmetic, residue generators, residue number system (RNS).**

I. Introduction

RESIDUE ARITHMETIC has been used in digital computing systems for various purposes for many years. Two most important applications of residue arithmetic are: 1) fault-tolerant digital systems protected against undetected errors using arithmetic error detecting and/or error-correcting codes [1]–[6]; and 2) high-performance digital signal processing (DSP) hardware which makes use of the parallel nature of the residue number systems (RNS's) arithmetic or number theoretic transforms [7]–[12]. Renewed interest in residue arithmetic is exemplified by two most recent commercial products: the large mainframe IBM Enterprise System/9000 using residue code mod 15 [6] and the INMOS' IMS A110 two-dimensional filter/convolver using RNS [7].

Manuscript received October 21, 1991; revised May 21, 1992 and June 10, 1993. A preliminary version of this paper was presented at the 10th Int. Symp. on Computer Arithmetic, Grenoble, France, June 23–28, 1991. This work was supported in part by the grant No. 3 P406 002 02 from Komitet Badań Naukowych, Poland.

The author is with the Technical University of Wrocław, Institute of Engineering Cybernetics, ul. Janiszewskiego 11/17, 50-372 Wrocław, POLAND.

IEEE Log Number 9212755.

Either application uses a *residue generator* (or *generator mod A*), i.e. a circuit that computes $|X|_A$, the residue modulo A, from a given binary number X. In a fault-tolerant digital system using an arithmetic error detecting code (EDC), it is used to build an encoding/decoding circuitry [1]–[4], whereas in an RNS-based digital system, a number of residue generators forms a binary-to-residue converter [9]–[16]. However, since in either application the generator is the overhead which can compromise the speed of the system, it should be a fast circuit realized with a minimum of hardware.

Consider a generator mod A with n inputs $\{x_{n-1}, \cdots, x_1, x_0$ and a outputs $\{s_{a-1}, \cdots, s_1, s_0\}$, where $a = \lceil \log A \rceil$ (by $\log A$ we always denote logarithm of A to the base 2). The generator converts an integer $X = \sum_{j=0}^{n-1} 2^j x_j$ into $|X|_A = \sum_{j=0}^{a-1} 2^j s_j$, i.e., it computes

$$|X|_A = \left| \sum_{j=0}^{n-1} 2^j x_j \right|_A . \tag{1}$$

The most obvious scheme of generating $|X|_A$ is a division of X by A to find the remainder, but this technique is very slow. More efficient is the algorithm from [9], based on the expression

$$|X|_A = \left| \sum_{j=0}^{n-1} |2^j|_A x_j \right|_A . \tag{2}$$

If the values of $|2^j|_A$ are directly available, $|X|_A$ may be computed by merely adding mod A those terms $|2^j|_A$ for which $x_j = 1$. The summation mod A of n residue can be performed by a tree of $n-1$ two-operand carry-propagate adders (CPA's) mod A, a regular but costly and time-consuming scheme. This approach was suggested to build universal area-time efficient VLSI binary-to-residue number system converters [15]. A modification of this method with reduced number of CPA's at the cost of ROM's, suitable for pipelining, was given in [16]. A bit-level systolic residue generator using n ROM's of size $2^a \times a$ was given in [13]. An alternative method for constructing the generator by using multistage ROM look-up tables was considered in [11]. All these methods aimed at RNS-based systems, where n is relatively small, seem unnecessarily complex both in terms of cost and speed to realize encoding and decoding circuitry for arithmetic EDC's, which may have $n = 112$ [6].

The generator mod $A = 2^a - 1$ needs a special consideration. Let $b = \lceil n/a \rceil$ be the number of a-bit bytes B_i into which an

TABLE I
LIST OF ALL MODULI A WITH $Per(A) = P(A) \leq 12$

$P(A)$	3	4	5	6	7	8	9	10	11	12
A	7	15	31	21, 63	127	51, 85, 255	73, 511	93, 341, 1023	23, 89, 2047	35, 39, 45, 91, 105, 117, 195, 273, 315, 455, 585, 819, 1365, 4095

TABLE II
LIST OF ALL MODULI A WITH $Per(A) = HP(A) \leq 12$

$HP(A)$	1	2	3	4	5	6	7	8	9	10	11	12
A	3	5	9	17	11, 33	13, 65	43, 129	257	19, 27, 57 171, 513	25, 41, 205, 1025	683, 2049	241, 4097

input n-tuple is partitioned. It was shown [1], [2] that due to

$$|2^{ta}|_{2^a-1} = 1, \qquad t \text{ nonnegative integer}, \tag{3}$$

it follows that

$$|X|_{2^a-1} = \left|\sum_{i=0}^{b-1} b_i 2^{ai}\right|_{2^a-1} = \left|\sum_{i=0}^{b-1} b_i\right|_{2^a-1}, \tag{4}$$

where b_i is the decimal value of B_i, $0 \leq b_i \leq 2^a - 1$, and the computation of $|X|_{2^a-1}$ can be accomplished with a tree of $b-1$ adders mod $2^a - 1$ (i.e. 1's complement adders). Since the generators available for $A \neq 2^a - 1$ have always been significantly more complex than for $A = 2^a - 1$, hence arithmetic EDC's with the check base $A = 2^a - 1$ have been called *low-cost arithmetic codes* [1], [2]. This scheme has been recommended for implementation of encoding/decoding algorithms for low-cost arithmetic codes in all the literature known to the author, including [1]–[3] (see [4] for many other references). Also, only low-cost arithmetic EDC's have been used in practical applications reported to date [1]–[3], [6]. However, recent work by the author [4] shows that efficient decoding algorithms for arithmetic EDC's with check bases other than $A = 2^a - 1$ can be derived by using the new design concepts which shall be presented in this paper.

A multioperand modular adder (MOMA) is another circuit that has been frequently used to build RNS hardware for various applications involving multiply-and-add intensive computations such as digital filtering, convolution, and FFT processing [10], and to implement a residue-to-binary converter based on the Chinese remainder theorem (CRT) [18], [21], [22]. Several MOMA's employing carry-save adders (CSAs) have been proposed [17]–[22]. Amongst word-serial designs the most efficient are those from: [17]—for $A = 2^a - 1$, [19]—for $A = 2^{a-1} + 1$, and [22]—for any other A. To date, only the design from [17] uses a CSA with end-around carry (EAC).

In this paper, we shall show how to design new residue generators and MOMA's for any A, which are faster and less complex than existing designs. Section II introduces the mathematical groundwork for the periodicity of the series of powers of 2 taken mod A: two basic congruential identities, known for $A = 2^a - 1$ and $2^{a-1} + 1$ only, are generalized to any odd A. This periodicity is a key idea for designing a general CSA/CPA network with EAC proposed in Section III. Section IV describes four new schemes of a parallel n-input generator mod A, suitable for various A and n, many of which can be pipelined on a full-adder level. New efficient parallel and word-serial architectures of the MOMA's for any A are given in Section V. Summary and applications of the results given here are presented in Section VI.

II. PERIODIC PROPERTIES OF $|2^j|_A$

The periodicity of the series of $|2^j|_A$ will be of key importance for designing new generators mod A and MOMA's proposed here. We start with two definitions adapted from [5].

Definition 1: The *period of the odd module* A $P(A)$ is the minimum distance between two distinct 1's in the sequence of residues of powers of 2 taken mod A, i.e. $P(A) = \min\{j | j > 0 \text{ and } |2^j|_A = 1\}$. Henceforth $P(A)$ will be simply called the *period of* A.

Definition 2: The *half-period of the odd module* A $HP(A)$ is the minimum distance between a pair of subsequent 1 and $A-1$ in the sequence of residues of powers of 2 taken mod A. (Note that $(A-1) \bmod A \equiv -1$.)

Whereas $P(A)$ exists for any A, $HP(A)$ exists for some A only. $HP(A)$ is called a half-period because if it exists then $P(A) = 2 \cdot HP(A)$. If $HP(A)$ does not exist it is assumed that $HP(A) = \infty$. $HP(A)$ and $P(A)$ can be calculated by using the following recursive equation

$$|2^i|_A = \left|2 \cdot |2^{i-1}|_A\right|_A. \tag{5}$$

However, for some A the following formulas are known to compute $HP(A)$ and $P(A)$ [5]:

- $HP(2^a - 1)$ does not exist whereas $P(2^a - 1) = a$;
- $HP(2^{a-1} + 1) = (a-1)$ and $P(2^{a-1} + 1) = 2(a-1)$;
- If $A = kB$ with k and B odd, then $P(A)$ is some multiple of $P(B)$.

In many cases the following single measure of periodicity of $|2^j|_A$ shall be useful

$$Per(A) = \min\{HP(A), P(A)\}. \tag{6}$$

All moduli with $Per(A) \leq 12$ are listed in Table I and II.

It can be shown that the following properties hold for any odd A.

TABLE III
THE MINIMUM NUMBER OF LEVELS $\theta(k)$ ON A CSA TREE THAT PROCESSES k INPUT OPERANDS [17]

k	3	4	$5 \div 6$	$7 \div 9$	$10 \div 13$	$14 \div 19$	$20 \div 28$	$29 \div 42$	$43 \div 63$
$\theta(k)$	1	2	3	4	5	6	7	8	9

Property 1: $a \leq P(A) \leq A - 1$.

Property 2: If $HP(A)$ exists, then $a - 1 \leq HP(A) \leq (A-1)/2$.

Property 3: $\text{Per}(A) \leq (A-1)/2$.

Now, with the concept of the period of A, we generalize the identity (3) to

$$|2^{tP(A)+j}|_A = |2^j|_A, \qquad t \text{ is any nonnegative integer.} \quad (7)$$

which holds for any A.

Similarly, given an odd A with $HP(A) < \infty$, with the concept of the half-period of A, we generalize the well known identity

$$|2^{t(a-1)}|_{2^{a-1}+1} = (-1)^t \quad (8)$$

to

$$|2^{t \cdot HP(A)+j}|_A = (-1)^t |2^j|_A. \quad (9)$$

III. CSA/CPA NETWORK WITH END-AROUND CARRY

CSAs have been used to implement MOMA's by many authors [17]–[22]. However, the MOMA for $A = 2^a - 1$ given in [17] (pp. 98–100) is the only circuit used in residue arithmetic that employs a *CSA/CPA network with EAC* that has been reported in the literature. Implementation of two-operand modulo adders using CPA's with EAC were given in [8].

For $A = 2^a - 1$ we have $P(2^a - 1) = a$ and the operation of an a-bit CSA/CPA network with EAC is justified by (3). For any odd A we may employ the $P(A)$ concept and a more general *$P(A)$-bit CSA/CPA network with EAC* whose operation is justified by (7). Suppose that we wish to find the sum mod A of $k \geq 3$ $P(A)$-bit numbers in two steps: first to find their sum using a CSA/CPA network, and then to perform the final conversion to the reside mod A with some final converter (FC). Note that in the CSA/CPA network, a FA with inputs of weight $|2^{P(A)-1}|_A$ generates the carry signal of weight $|2 \cdot 2^{P(A)-1}|_A = 1$. Thus, it can be treated as the EAC and allows to keep intermediate results of summation within the range of $P(A)$ (rather than $P(A) + \lceil \log k \rceil$) bits at no cost. A remarkable feature of this CSA/CPA network is the cyclic nature of addition mod A with the cycle length $P(A)$, which allows any $P(A)$-bit CPA with EAC to begin and end operation at *any* point of cycle (see Fig. 3).

It should be noted that the idea of grouping the bits of the same weight $|2^i|_A$ was used by Zhang *et al.* [14] in a neural-like network implementations of a variety of RNS hardware, including a binary-to-residue converter, but no specific logic implementation was detailed. The $P(A)$ concept of periodicity of $|2^i|_A$ used here actually represents the same idea, but it is presented in a more general framework and it is extended to include $HP(A)$.

IV. NEW GENERATORS MOD A

In this section, we propose four new efficient parallel architectures of the n-input generator mod A which are suited to cover the whole spectrum of A and n. It shall be shown that they can be pipelined with the pipeline registers introduced after each FA.

Throughout this paper we shall use the following notation and estimations of the delay introduced by various circuits: t_{FA}—the delay of a FA, t_L—the delay of a latch, $t_{CPA(p)}$—the delay of a p-bit CPA, t_{Count}—the delay of a counter, $t_{Conv(n)}$—the delay of an n-input a-output converter (which is $O(\log n)$ if implemented using ROM). Similarly as in [21], the following delays in terms of NAND gate delay as a unit are assumed: $t_{NOT} = t_{NAND} = t_{NOR} = 1$, $t_{FA} = t_{MUX} = 2$, and $t_{Conv(n)} = 3$ for $n \leq 4$. Finally, unlike in [21], we assume that the fastest p-bit CPA can be implemented:

- for small p—as a ripple-carry adder employing the FA module proposed recently in [23]; the complexity of the latter is that of a standard FA but it introduces the delay of a single three-input NAND gate for carry propagation, i.e. $t_{CPA(p)} = p$;
- for large p—as a carry-lookahead adder (CLA), with $t_{CLA(p)} = 12$ for $p \leq 64$ [17].

$\theta(k)$, the minimum number of levels in the CSA tree with k operands, can be found in Table III. In [22] it was proved that $\theta(k) = O(\log k)$.

A. CPA-Based Generator Mod A

The primary goal of this paper is to perform the computation of $|X|_A$ or multioperand addition mod A using CSA's. However, all CSA-based designs that shall be given here, require the minimum width of a CSA to be min $\{a, Per(A)\}$. Thus, there are various n and A, which preclude using a CSA, but still allow to use efficiently a CPA. (Obviously, for small n, say $n \leq 10$, the best implementation could be one custom-designed $2^n \times a$ ROM look-up table.)

The new special scheme given in Fig. 2 works as follows. The set of n input bits is partitioned into two disjoint subsets $X_1 = \{x_{n-1}, \cdots, x_a, x_{a-1}\}$ and $X_2 = \{x_{a-2}, \cdots, x_1, x_0\}$, so that $R_2 = \sum_{i=0}^{a-2} x_i 2^i$ is a residue mod A. Let

$$R_1 = |x_{a-1}2^{a-1} + x_a 2^a + \cdots + x_{n-1}2^{n-1} + COR(n, A)|_A \quad (10)$$

where $COR(n, A) \neq 0$, if and only if the circuit is used as a FC for some larger generator mod A or a (k, A) MOMA, either based on the $HP(A)$ concept (see Sections IV-C, V-A, and V-B), and X_1 is converted in parallel to $|X_1'|_A = R_1$ and $X_1'' = R_1 - A$, where X_1'' is in the 2's complement form. Then

TABLE IV
PERFORMANCE COMPARISON OF PIPELINED RESIDUE GENERATORS

Generator	Hardware		Delay	
	FA's	ROM Size	Pipeline Stages	Pipeline Interval
$A = 2^a - 1$	$n - a$	—	$\theta\left(\lceil \frac{n}{a} \rceil\right) + a$	t_{FA}
$A = 2^{a-1} + 1$	n	—	$\theta\left(\left\lceil \frac{n}{a-1} \right\rceil\right) + a + 2$	$\max\{t_{FA}, t_{MUX}\}$
$Per(A)$-Based	$n - Per(A) + 2a$	$2^{Per(A)-a+2} \times 2a$	$\theta\left(\left\lceil \frac{n}{Per(A)} \right\rceil\right) + 2a + 2$	$\max\{t_{Conv(Per(A)-a+1)}, t_{FA}, t_{MUX}\}$
CPA-Based	$2a$	$2^{n-a+1} \times 2a$	$a + 2$	$\max\{t_{Conv(n-a+1)}, t_{FA}, t_{MUX}\}$
MOMA-Based	$(k_1 + 2)a - b^{(1),(2)}$	$k_1(2^r \times a) + 2^{b+1} \times a$	$\theta\left(\left\lceil \frac{n}{Per(A)} \right\rceil\right) + \theta(k_1 + 1) + 2a + 3^{(3)}$	$\max\{t_{FA}, t_{Conv(r)}, t_{Conv(b+1)}, t_{MUX}\}$
[12]	—	$n(2^a \times a)$	n	$t_{ROM}(2^a \times a)$
[16]	$(k_3 - 1)2a^{(4)}$	$k_3(2^q \times a)$	k_3	$t_{ROM}(2^q \times a) + 2t_{CPA(a)} + t_{MUX}$

Note: 1) $k_1 = \lceil n * /r \rceil = O(n)$, r = const; 2) $b = \lceil \log_2(k_1 + 1) \rceil = O(\log n)$; 3) For $x < 3\theta(x) = 0$; and 4) $k_3 = \lceil n/q \rceil = O(n)$, q = const.

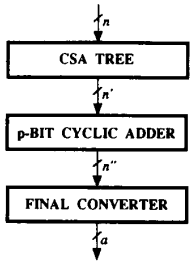

Fig. 1. General structure of the new n-input generator mod A and the k-operand adder mod A.

$|X|_A$ is computed according to

$$|X|_A = \begin{cases} |X_1'|A + X_2, & \text{if } c''_{\text{out}} = 0, \\ X_1'' + X_2, & \text{if } c''_{\text{out}} = 1, \end{cases} \tag{11}$$

by taking into account a least significant bits (LSB's) only. This approach is justified by: 1) $-A = |-A|_{2^a} = 2^a - A$; and 2) $0 \le |X_1'|_A \le A - 1$ implies $2^a - A \le X_1'' \le 2^a - 1$.

The above circuit can be used as a stand-alone n-input generator mod A for some n such that $10 < n \le \min\{2Per(A), a + 10\}$ or as the final converter in other generators and MOMA's. For moderate $n - a$, say $n - a \le 3$, it can be pipelined on a FA level (see Table IV).

B. Generator Mod A Based on the P(A) Concept

Suppose that $Per(A) = P(A)$ (see Table I) and the ratio $v = \lceil n/P(A) \rceil$ is sufficiently large. Equation (7) implies that for $n > P(A)$ the bits $x_j, x_{P(A)+j}, x_{2P(A)+j}, \cdots$ $(0 \le i \cdot P(A) + j < n)$ represent the same residue $|2^j|_A, 0 \le j \le P(A) - 1$. This allows to partition n input bits into $P(A)$ subsets $G_j = \{x_q \le q = tP(A) + j$ and $0 \le q \le n-1\}, j = \{0, 1, 2, \cdots, P(A)-1\}$. The sets G_j are cyclically ordered according to increasing indices j taken mod $P(A)$; in particular, the sets $G_{P(A)-1}$ and G_0 are adjacent. Now a new n-input generator mod A, shown in Fig. 1, can be designed according to the following procedure.

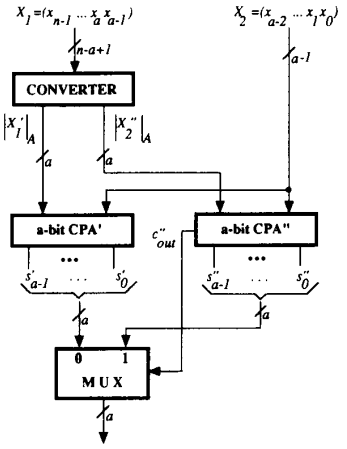

Fig. 2. CPA-based generator mod A, usually used as the final converter.

PROCEDURE 1

Step 1: Partition the set of input bits $\{x_0, x_1, \cdots, x_{n-1}\}$ into subsets $G_j = \{x_q | q = tP(A) + j\}, 0 \le j \le P(A) - 1$.

Step 2: If $n \le 2P(A) + 1$ assume $n' = n$ and go to Step 3. Otherwise, compress n input bits to n' using a CSA with EAC of length up to $P(A)$. For $n' = 2P(A)$ any set G_j has two bits, whereas for $n' = 2P(A) + 1$ one set, say G_{j^*}, has three bits while all other sets G_j have two bits.

Step 3: Compress n' bits to $n'' = P(A) + 1$ by using a p-bit CPA with EAC.

a) If $n' = n$ then $p = n' - P(A)$ and the CPA starts with G_0;
b) If $n' = 2P(A)$ then $p = P(A)$ and the CPA can start with any G_j;
c) If $n' = 2P(A) + 1$ then $p = P(A)$ and the CPA starts with G_{j^*}.
d) In any case, at the end of the addition every set G_j except for one, say G_r, contains a single bit, i.e., $G_j = \{y_j\}$ for $j \ne r$ and $G_r = \{y_r', y_r''\}$. Depending on the case, r equals: a) $n' - P(A)$; b) $j + 1$; or c) $j^* + 1$.

Step 4: Compute $|y_0 2^0 + y_1 2^1 + \cdots + y_r' 2^r + y_r'' 2^r + \cdots + y_{P(A)-1} 2^{P(A)-1}|_A$. □

G5	G4	G3	G2	G1	G0	
5	5	5	5	6	6	
FA	FA	FA	FA	2 FA	2 FA	STAGE 1
4	4	4	5	4	3	
FA	FA	FA	FA	FA	FA	STAGE 2
3	3	3	4	3	2	
FA	FA	FA	FA	FA	HA	STAGE 3
2	2	2	3	2	2	
←FA—	←FA—	←FA—	←FA	←FA—	←FA—	6-BIT CPA WITH EAC
1	1	1	2	1	1	
y_5	y_4	y_3	y'_2, y''_2	y_1	y_0	

Fig. 3. CSA/CPA network for the 32-bit generator mod 21.

Step 4 is executed by the FC realized as a CPA-based generator mod A, except $A = 2^a - 1$ when it is not needed because $n' = a$.

Example 1: Fig. 3 shows a shorthand description of the CSA/CPA network for the 32-bit generator mod 21. The input bits are partitioned into $P(21) = 6$ sets $G_0 = \{x_0, x_6, x_{12}, x_{13}, x_{24}, x_{30}\}, \cdots$, and $G_5 = \{x_5, x_{11}, x_{17}, x_{23}, x_{29}\}$. The contents of a column G_j of the table alternately either shows how many bits of residue weight $|2^j|_{21}$ are present at a given stage of computation or specifies the number of FA's and half-adders (HA's) that operate on the bits from a given set G_j. The FC which computes $|1 \cdot y_0 + 2 \cdot y_1 + 4 \cdot y'_2 + 4 \cdot y''_2 + 8 \cdot y_3 + 16 \cdot y_4 + 11 \cdot y_5|_{21}$ can be realized with a ROM look-up table of size 128·5 bits. Two bits y'_2, y''_2 represent the *Sum* output of the bottommost FA in the column G_2 and the *Carry* output of the FA in the column G_1. □

Table I shows that Procedure 1 is naturally well suited for any $A = 2^a - 1$ (including $A = 3$) which do not use FC, but it is also suitable for some other A for which $Per(A) = P(A)$ is reasonably small. The upper-bounds for complexity estimation of the pipelined version of the $P(A)$-based generator can be found in Table IV.

C. Generator Mod A Based on the HP(A) Concept

If $Per(A) = HP(A)$ (see Table II) and the ratio $v = \lceil n/HP(A) \rceil$ is sufficiently large, the generator is designed using similar concepts as the $P(A)$-based generator and has the same general structure as shown in Fig. 1.

Let n input bits be partitioned into $v = \lceil n/HP(A) \rceil$ $HP(A)$-bit bytes $B_0, B_1 \cdots B_{v-1}$, starting with the LSB; obviously, B_{v-1} may be incomplete, i.e., it may consist of only d MSB's of $X, 0 < d < HP(A)$. For b_i, the decimal value of B_i, we have $0 \le b_i < 2^{HP(A)}$. Therefore,

$$|X|_A = \left| \sum_{i=0}^{v-1} b_i \cdot 2^{i \cdot HP(A)} \right|_A . \tag{12}$$

Due to (9), the computation of (12) can first be simplified to

$$|X|_A = \left| \sum_{i=0}^{v-1} (-1)^i b_i \right|_A . \tag{13}$$

G3	G2	G1	G0	
—	5	5	6	
—	FA	FA	2 FA	STAGE 1
—	4	5	3	
—	FA	FA	FA	STAGE 2
—	3	4	2	
—	FA	FA	—	STAGE 3
—	2	2	3	
—	←FA—	←FA—	←FA	3-BIT CPA
1	1	1	1	
y_3	y_2	y_1	y_0	

Fig. 4. CSA/CPA network for the 16-bit generator mod 9.

Then, because $|2^{HP(A)}|_A = -1$ implies $|-b_i|_A = |(2^{HP(A)} - 1 - b_i) + (1 - 2^{HP(A)})|_A = |\overline{b_i} + 2|_A$, we receive

$$|X|_A = \left| \sum_{i=0, i \text{ even}}^{v-1} b_i + \sum_{i=0, i \text{ odd}}^{v-1} \overline{b_i} + 2\lfloor v/2 \rfloor \right|_A , \tag{14}$$

where $\overline{b_i}$ denotes the decimal value of the bit-by-bit complement of B_i. 1's complement

Finally, the computation of (12) using an $HP(A)$-bit CSA/CPA network with complemented EAC is justified by

$$|X|_A = \left| \sum_{i=0, i \text{ even}}^{v-1} b_i + \sum_{i=0, i \text{ odd}}^{v-1} \overline{b_i} + COR(n, A) \right|_A \tag{15}$$

where

$$COR(n, A) = |2 \cdot \lfloor v/2 \rfloor + w - l|_A \tag{16}$$

is the total correction—a constant integer characteristic for a CSA/CPA network used. Three terms in (16) occur, respectively, due to the following: (1) $\lfloor v/2 \rfloor$ odd-numbered complemented bytes; (2) $w \neq 0$ and $w = \sum_{i=d}^{HP(A)-1} 2^i$ iff B_{v-1} with even v is incomplete—because B_{v-1} has $HP(A) - d$ leading 0's which become 1's after complementing the whole byte B_{v-1}; and (3) l complemented EAC's used by the CSA/CPA network—because each EAC bit generated by an FA or HA operating on the bits from $G_{HP(A)-1}$ is complemented and directed back to G_0, what requires a correction by $|2 + \sum_{i=1}^{HP(A)-1} 2^i|_A = |2^{HP(A)}|_A = -1$.

Example 2: Design a parallel 16-input generator mod 9. Since $HP(9) = 3$ the input bits are first partitioned into $v = 6$ 3-bit bytes $B_0 = \{x_2, x_1, x_0\}$, $B_1 = \{x_5, x_4, x_3\}, \cdots$, $B_5 = \{x_{15}\}$ with the bits from B_1, B_3, and B_5 complemented. Then, they are assigned to the sets G_0, G_1, and G_2. The CSA/CPA network is given in Fig. 4. No correction is needed due to $COR(16, 9) = |6 + 6 - 3|_9 = 0$. If unbiased residue mod 9 is needed, a simplified FC is used, which computes $|1 \cdot y_0 + 2 \cdot y_1 + 4 \cdot y_2 + 8 \cdot y_3|_9$ (it consists of a subtractor of 9 and a multiplexer). (A viable alternative is to replace both the CPA and the FC with a 128 × 4 ROM.)

The whole generator can be pipelined on the FA level. The generator with the smallest latency may have only 9 pipeline stages, provided that the basic CPA and the subtractor operate

in parallel, with the latter delayed by one clock cycle. Its hardware cost is upper-bounded by 16 FAs, 4 HA's, and 65 latches. Similar pipelined generator from [12] uses 16 stages of $16 \cdot 9$ bits ROM cells and the clock cycle determined by the ROM cycle-time, and it is clearly inferior to our design. □

A formal procedure for designing the $HP(A)$-based generator can be derived by modifying Procedure 1 in the following way: 1) replace $P(A)$ with $HP(A)$; 2) complement input bits from odd-numbered bytes and all EAC's; and 3) add $COR(n, A)$ to derive the final result. Table II shows that this generator is naturally well suited for any $A = 2^{a-1} + 1$ and many other A. The upper-bounds for complexity estimation of the pipelined version of the $HP(A)$-based generator can be found in Table IV.

Finally, it is worth to note that the residue generator mod $A = 2^{a-1} + 1$ uses a significantly simplified FC compared to the scheme from Fig. 2. Also, because an intermediate result produced by the CSA/CPA network occupies exactly a bits, no FC may be used in some applications, provided that extracting unbiased residue can be postponed until all residues in the range $[0, 2^{a-1}]$ are actually needed.

D. Generator Mod A Using a MOMA

Here, we propose the most general residue generator which handles those A and n for which three previous schemes are inefficient.

Example 3: Design the 32-input generator mod 49. Since $Per(49) = P(49) = 21$, we cannot take advantage of periodicity, since $|G_j| = 2$ for $0 \leq j \leq 10$ and $|G_j| = 1$ for $11 \leq j \leq 20$ ($|G_j|$ denotes the cardinality of the set G_j). A more efficient generator can be built by using a 6-operand MOMA mod 49 (for short $(6, 49)$ MOMA). Twelve bits from the sets $G_j, 0 \leq j \leq 5$, are treated as two operands (the fact that they may be residues mod 49 biased by $+49$ is immaterial here, but it may matter when the FC for a MOMA is concerned). Remaining $n^* = 20$ bits from the sets G_j, $6 \leq j \leq 20$ can be converted to four residues mod 49 with small ROM's. Now these six 6-bit numbers are added by a $(6, 49)$ MOMA designed by using Procedure 3 given in Section V. It can be shown that the fully-pipelined version of this generator has 17 stages and $t_{CL} = \max\{t_{Conv(5)}, t_{FA} + t_L, t_{MUX}\} \simeq 3$. □

Example 4: Design the 32-input generator mod 19. Since $Per(19) = HP(19) = 9$ the input bits are partitioned onto $v = 4$ 9-bit bytes $B_0 = \{x_8, \cdots, x_0\}$, $B_1 = \{x_{17}, \cdots, x_9\}$, $B_2 = \{x_{26}, \cdots, x_{18}\}$, and $B_3 = \{x_{31}, \cdots, x_{27}\}$, where, according to (15), B_1 and B_3 are bit-by-bit complemented. Now two stages of a CSA with one complemented EAC (using a total of 14 FA's) reduce the bits from $HP(19) = 9$ sets G_i. Then, the MOMA-based generator initially uses two ROM's of size 16×5 and 32×5 bits to convert $n^* = 9$ bits from the sets G_2, G_6, G_7, and G_8 to two residues mod 19. Finally, a $(4, 19)$ MOMA is applied. The total correction $COR(32, 19) = |481|_{19} = 6$ can be added by any ROM used. □

PROCEDURE 2

Step 1: Find $m = \min\{HP(A), P(A), n\}$. If $n > HP(A)$ first form $v = \lceil n/HP(A) \rceil HP(A)$–bit bytes B_i of n input bits and bit-by-bit complement odd-numbered bytes B_i, $0 \leq i \leq v - 1$. Partition n input bits onto sets $G_j = \{x *_q | q = t \cdot m + j\}$, $0 \leq j \leq m - 1$, where $x*_q$ denotes: $\overline{x}_q$—if $n > HP(A)$ (i.e., x_q was assigned to B_i with odd i), and x_q—otherwise.

Step 2: If $|G_a| \geq 3$ compress n bits using CSA until at most one set G_j has three bits.

Step 3: Partition arbitrarily $n* = \sum_{j=a}^{m-1} |G_j|$ bits from the sets G_j, $a \leq j \leq m - 1$, onto some k_1 subsets and design k_1 converters that convert them into residues mod A and if $n > HP(A)$ add $COR(n, A)$.

Step 4: Compute $|X|_A$ using the $(k_1 + k_2, A)$ MOMA, where $k_2 = \max\{|G_j|, 0 \leq j \leq a - 1\}$.

Table IV shows that proper selection of k_1 (in Step 3) has crucial impact on performance of the whole generator.

E. Complexity Estimation and Comparison

Table IV provides upper bounds on the amount of hardware and time delay of fully pipelined versions of the new generators and compares them against pipelined designs from [12] and [16]. It is assumed that new generators that require a FC employ one from Fig. 2 (or its simplified version for $A = 2^{a-1} + 1$) and, for the MOMA-based generator, the worst case of A and n is considered—when $Per(A) \geq n$. Obviously, faster and less complex versions can easily be obtained for many A and n, e.g., by replacing ripple-carry adders with faster CPA's and by replacing the FC from Fig. 2 or both the CPA and the FC with one ROM look-up table.

From Table IV we can draw the following corollaries regarding new residue generators.

Corollary 1: They all can be build using about n FA's and only the MOMA-based generator also requires $O(n)$ ROM bits.

Corollary 2: They have at most $O(\log n)$ pipeline stages.

Corollary 3: They have the pipeline interval $O(1)$ except the MOMA-based generator for which it is $O(\log n)$.

Corollary 4: If double representation of residues mod $2^{a-1} + 1$ is allowed, the new generator mod $2^{a-1} + 1$ has similar performance as the new generator mod $2^a - 1$.

The generators from [12] and [16] use $O(n)$ ROM bits, whereas the latter also uses $O(n)$ FAs, i.e. they have similar hardware complexity as our MOMA-based design (which is the least efficient amongst our designs). However, they are generally more complex, since they are build using larger blocks than our generators. As for latency, the designs form [12] and [16] have $O(n)$ pipeline stages compared to our $O(\log n)$. Similarly as our periodicity-based generators, they have the pipeline interval $O(1)$ but, again, they involve larger constants. Only the MOMA-based design has the pipeline interval $O(\log n)$, but the constant r can be selected to optimize pipeline interval (i.e., to keep $t_{Conv(r)} \simeq t_{Conv(b+1)}$) and make it close to $t_{ROM}(2^a \times a)$ in [12]. Then, however, the signal latency of two existing designs with $O(n)$ pipeline stages is significantly larger. Overall, the new generators offer significantly better throughput than existing designs with the

TABLE V
LIST OF ALL $A \neq \{2^a - 1, 2^{a-1} + 1\}$ FOR WHICH $k_c(A) \leq 9$

A	11	13	21	43	51	57	73	85	171	205	341	585	681	819
$Per(A)$	5	6	6	7	8	9	9	8	9	10	10	12	12	12
k_c	4	6	4	4	6	9	8	4	4	6	4	8	4	6

least cost, especially when the moduli with small $Per(A)$ are concerned.

V. MULTIOPERAND ADDER MOD A

Let

$$X_0 = (x_{a-1}, \cdots, x_1, x_0)$$
$$X_1 = (x_{2a-1}, \cdots, x_{a+1}, x_1), \cdots, \text{ and}$$
$$x_{k-1} = (x_{ka-1}, \cdots, x_{(k-1)a})$$

be k residues mod A. A k-operand adder mod A, which will be henceforth referred to as a (k, A) MOMA, is a circuit that computes

$$|X|_A = \left| \sum_{i=0}^{k-1} X_i \right|_A \tag{17}$$

which is equivalent to

$$|X|_A = \left| \sum_{j=0}^{a-1} |2^j|_A \cdot \left(\sum_{i=0}^{k-1} x_{ia+j} \right) \right|_A . \tag{18}$$

Any new (k, A) MOMA proposed here first computes

$$\left| \sum_{j=0}^{a-1} |2^j|_A \cdot \left(\sum_{i=0}^{k-1} x_{ia+j} \right) \right|_{2^{Per(A)}-1} \tag{19}$$

using a CSA/CPA network with EAC, and then it performs the final evaluation of

$$|X|_A = \left| \left| \sum_{j=0}^{a-1} |2^j|_A \cdot \left(\sum_{i=0}^{k-1} x_{ia+j} \right) \right|_{2^{Per(A)}-1} \right|_A . \tag{20}$$

It employs similar ideas and structure as the generator mod A from Fig. 1. The maximum number of sets G_j which are used in the CSA/CPA network of the new MOMA is given by

$$q = \min\{Per(A), m\}, \tag{21}$$

where

$$m = \lceil \log[k(A-1)+1] \rceil \tag{22}$$

is the number of bits needed to encode the largest number resulting from the summation of k residues mod A in binary.

Since, a word-serial MOMA can be obtained from a parallel MOMA by simple replacement of a CSA/CPA network only form some k and A, they shall be considered separately.

A. Parallel MOMA

1) $A \neq 2^{a-1} + 1$ PROCEDURE 3

Step 1: Find $m = \lceil \log[k(A-1)+1] \rceil$.
Step 2: Partition $k \cdot a$ input bits onto a subsets $G_j = \{x_{va+j} | 0 \leq v \leq k-1\}$ for $0 \leq j \leq a-1$, and assume $G_j = \varnothing$ for $a \leq j \leq q-1$.
Step 3: Compress $k \cdot a$ input bits to n' by using the CSAs with EAC of length up to q until at most one set, say G_j, has three bits.
Step 4: Compress n' bits to n'' by using a p-bit CPA with EAC; $n'' = q$ when no EAC occurs, and $n'' = q+1$ otherwise.
Step 5: Compute $|y_0 2^0 + y_1 2^1 + \cdots + y'_r 2^r + y''_r 2^r + \cdots + y_{q-1} 2^{q-1} + COR_P(k, A)|_A$, where $COR_P(k, A)$ is the total correction needed by a *parallel* MOMA with l EAC's. □

Notes

1) Three different types of CSA/CPA networks are used in MOMA's designed by Procedure 3:
   a) without EAC—for $q = m < Per(A)$; $COR_P(k, A) = 0$;
   b) with complemented EAC—for $q = HP(A) < m$; $COR_P(k, A) = |-l|_A$ and each EAC bit produced by a CSA/CPA network is taken complemented; and
   c) with uncomplemented EAC—for $q = P(A) < m$; $COR_P(k, A) = 0$.
2) Any set G_j except for at most one, say G_r, produced in Step 4 contains a single bit y_j of weight $|2^j|_A$, $j \neq r$, whereas G_r may contain two bits $\{y'_r, y''_r\}$.
3) Step 5 is executed either by a ROM look-up table or the FC from Fig. 2. Also, if n' is small, say $n' \leq 10$, then one ROM look-up table may replace both the CPA and the FC.
4) The analysis of many examples showed that if HA's are cleverly used then the length of the CPA used in a parallel MOMA can be limited to $p \leq a$.

Note that the cyclic mode of operation of the CSA/CPA network for a (k, A) MOMA designed by Procedure 3 occurs for $A = 2^a - 1$ and $A = 2^{a-1} + 1$ (it is shown below) for any $k \geq 2$, whereas for other A, it occurs for a sufficiently large $k \geq k_c(A)$ (see Table V), where

$$k_c(A) = \lceil 2^{Per(A)} / (A-1) \rceil . \tag{23}$$

Thus, using the CSA/CPA network with EAC is a cost-free method of limiting n'': for any A specified in Table V and $A = 2^{a-1} + 1$ with $a \leq 10$ we have $n'' \leq 10$, which makes feasible implementing the FC with a single ROM.

2) $A = 2^{a-1} + 1$

For $A = 2^{a-1} + 1$ we have $HP(2^{a-1} + 1) = a - 1$, which is the only case of $Per(A) < a$. The weight $|2^{a-1}|_{2^{a-1}+1}$ of x_{ra+a-1}, the MSB of X_r is congruent to -1, and its complement has weight 1, provided that the correction by $|2^{HP(A)}|_{2^{a-1}+1} \equiv |2^{a-1}|_{2^{a-1}+1} \equiv -1$ is included in $COR_P(k, 2^{a-1} + 1)$. Overall, the design of a $(k, 2^{a-1} + 1)$ parallel MOMA closely resembles the design of the generator mod $2^{a-1} + 1$.

PROCEDURE 4

Step 1: Partition $k \cdot a$ input bits onto $a - 1$ subsets: $G_0 = \{x_{ra}, \overline{x}_{ra+a-1} | 0 \le r \le k - 1\}, G_j = \{x_{ra+j} | 0 \le r \le k - 1\}$ for $1 \le j \le a - 2$, and assume $G_{a-1} = \emptyset$.

Step 2: Compress $k \cdot a$ input bits to n' by using the CSAs (with l complemented EAC lines) of length up to $a - 1$ until at most one set (G_0 only) has three bits.

Step 3: Compress n' bits to a by using an $(a - 1)$-bit CPA.

Step 4: Compute $|y_0 2^0 + y_1 2^1 + \cdots + \cdots + y_{a-1} 2^{a-1} + COR_P(k, 2^{a-1} + 1)|_A$, where $COR_P(k, 2^{a-1} + 1) = |-(l + k)|_{2^{a-1}+1}$. □

Note that this approach provides an efficient 2-operand adder mod $2^{a-1} + 1$ as well.

B. Word-Serial (k, A) MOMA

Here, we shall explicitly consider only the design of a word-serial (k, A) MOMA using the CSA with EAC. A word-serial MOMA without EAC can be easily designed by slight modifications of the parallel version: the word-serial MOMA employs one-stage m-bit CSA followed by an m-bit CPA, where m is given by (22). It can also be obtained from a word-serial MOMA with EAC by removing some HA's operating on the MSB's and breaking the EAC line.

1) $A \neq 2^{a-1} + 1$

PROCEDURE 5

Let $S = (S_{Per(A)-1}, \cdots, C_1, C_0)$, $C = (C_{Per(A)-1}, \cdots, C_1, C_0)$ denote two registers which store the *Sum* and *Carry* outputs of the $Per(A)$-bit CSA.

Step 1: LOAD $S{:=}\ X_0$; LOAD $C{:=}\ \|2^{-1}|_A \cdot COR_S(k, A)|_A$.

Step 2: For $i = 1$ *to* $k - 1$ *do* $C + S{:=}(C + S + X_i)$ mod $(2^{\mathrm{Per(A)}} - 1)$.

Step 3: $Y{:=}C + S$, where: $Y = (y_{Per(A)}, y_{Per(A)-1}, \cdots, y_1, y_0)$, $y_i \in G_i$ for $0 \le i \le Per(A) - 1$, and $y_{Per}(A)$ is the carry generated by the CPA.

Step 4: $(s_{a-1}, \cdots, s_1, s_0){:=}Y$ mod A. □

The word-serial MOMA designed by the above procedure preserves the general structure from Fig. 1, but the CSA consists of only one stage with a input lines. During Step 1 the total correction $COR_S(k, A) = |-k|_A$ is taken into account, but only if the actual implementation of the MOMA uses complemented EAC due to $HP(A)$; otherwise $COR_S(k, A) = 0$. To allow dynamic changes of k, $COR_S(k, A)$ can be read from a look-up table.

2) $A = 2^{a-1} + 1$

Basically, a word-serial $(k, 2^{a-1} + 1)$ MOMA can be designed using Procedure 5, but it must use a CSA of length $P(2^{a-1} + 1) = 2(a - 1)$. Due to $Per(2^{a-1} + 1) < a$, a one-stage $(a - 1)$-bit CSA with complemented EAC cannot be used as in a parallel version, because the CSA would receive four bits of weight 1. Consequently, the condition for the occurrence of EAC changes from $k_c(2^{a-1} + 1) = 2$—for a parallel MOMA, to $k_{cS}(2^{a-1} + 1) = \lceil 2^{2(a-1)}/2^{a-1} \rceil = 2^{a-1}$—for a word-serial MOMA.

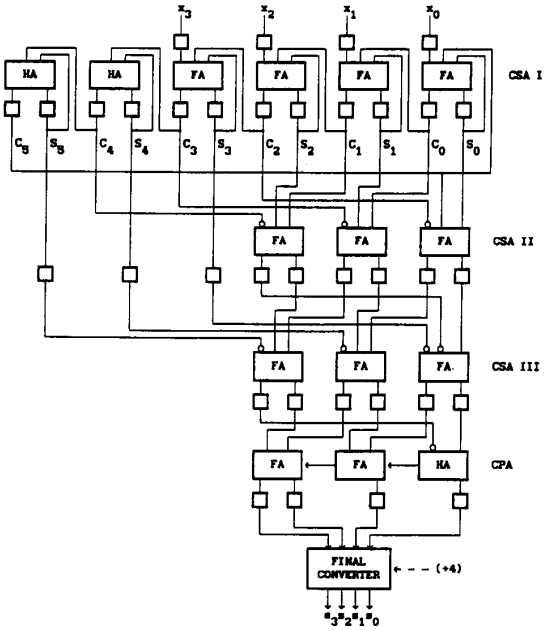

Fig. 5. Word-serial $(k, 9)$ MOMA.

The word-serial $(k, 2^{a-1} + 1)$ MOMA has the structure as shown in Fig. 5. It consists of three stages of the CSA: the $2(a - 1)$-bit basic CSA I with a input lines, and two $(a - 1)$-bit CSAs II and III. Remaining part is the same as in a parallel MOMA. It requires the constant correction $COR_S(k, 2^{a-1} + 1) = 4$ with terms contributed only by the CSAs II and III.

C. Performance Evaluation and Comparison

Table VI summarizes complexity characteristics of various MOMA's. The MOMA from [17]—the most efficient scheme known for $A = 2^a - 1$—is a special case of Procedure 3; it can be easily implemented as word-serial, parallel, or pipelined at the FA level. For other A, the word-serial MOMA's were considered in [18]–[20], whereas the fastest parallel architecture (which can be readily made word-serial) was given by Lee *et al.* [21], [22]. We don't consider the designs from [18] which were shown [20], [21] to be less efficient.

The complexity formulas given in Table VI were derived under the following assumptions.

1) Any new MOMA that requires an FC employs the scheme from Fig. 2.
2) For $A = 2^a - 1$ the output generated by the CPA with EAC stabilizes after executing one loop, i.e., it equals $t_{CPA(a)}$ which is a lower-bound.

TABLE VI
COMPLEXITY OF VARIOUS MOMA'S

MOMA	A	Hardware		Delay
		FA's	Other	
Parallel	2^a-1 [17]	$(k-1)a$	—	$\theta(k)t_{FA}+t_{CPA(a)}$
	$2^{a-1}+1$	$ka-1$	MUX	$[\theta(k+1)]t_{FA}+t_{CPA(a-1)}+t_{CPA(a)}+t_{MUX}$
	Other	$(k+1)a$	$<a$ HAs, $2^{\log k}\times 2a$ ROM, MUX	$\theta(k)t_{FA}+t_{CPA(a)}+t_{Conv(\log k)}+t_{CPA(a)}+t_{MUX}$
Word-Serial	2^a-1 [17]	$2a$	—	$(k-1)t_{FA}+t_{CPA(a)}$
	$2^{a-1}+1$	$5(a-1)$	a HAs, MUX	$(k+1)t_{FA}+t_{CPA(a-1)}+t_{CPA(a)}+t_{MUX}$
	Other	$3a+2\lceil\log k\rceil$	$2^{\log k+1}\times a$ ROM, MUX	$(k-1)t_{FA}+t_{CPA(a+\log k)}+t_{Conv(\log k+1)}+t_{CPA(a)}+t_{MUX}$
[19]	$2^{a-1}+1$	$3a$	Counter mod $(2^{a-1}+1)$, $2^{a+1}\times a$ ROM	$(k-1)(t_{FA}+t_{Count})+(t_{CPA(a)}+t_{Count})+t_{CPA(a)}+t_{Conv(a+1)}$
[20]	Any	$4a+12$	MUX	$(k-1)(4t_{FA}+t_{MUX})+t_{CPA(a+2)}+t_{MUX}$

3) For $A=2^{a-1}+1$ the formulas are derived for $k\geq 2^{a-1}$, i.e. when a cycle occurs, and thus are upper bounds for our MOMA's.
4) For $A\neq\{2^a-1, 2^{a-1}+1\}$ the worst case of A and k is considered—when EAC is not used (even when it could be used).
5) For any word-serial MOMA it is assumed that: a) the first operand X_0 is loaded to the *Sum* register of a CSA during the first cycle; and b) t_L is negligible.

Table VI shows that any new parallel (k, A) MOMA uses about $k\cdot a$ FA's and a small amount of extra hardware. The new word-serial MOMA's can be built using from $2a$ to $5(a-1)$ or $3a+2\lceil\log k\rceil$ FA's. Hardware estimations for $A=2^{a-1}+1$ are loose upper-bounds, so the reader may get an impression that the general MOMA is less complex. Actually, the special $(k, 2^{a-1}+1)$ MOMA uses less FA's than the general MOMA: for any k—if $a\leq 8$, and for larger k—if $a>8$, and for larger k—if $a>8$, and it is significantly faster for any k.

Compared to the word-serial MOMA from [20], our word-serial MOMA is less complex in most cases and significantly faster for any case. As for the new $(k, 2^{a-1}+1)$ MOMA's, either version introduces less delay than the CSA/CPA part alone of the MOMA from [19]. Since the latter uses a FC with one input more than our MOMA, it is also more complex for many a. An important advantage of our MOMA is a highly regular structure composed on only FA's and HA's which, unlike one from [19], can be easily clocked with $t_{CL}\simeq t_{FA}+t_L$. Finally, the partial result generated by the CSA/CPA network of our $(k, 2^{a-1}+1)$ MOMA $(k\geq 2)$ fits exactly in a bits and may not need a FC, provided that a result biased by $+(2^{a-1}+1)$ is acceptable, and then its overall performance becomes comparable to the most efficient known $(k, 2^a-1)$ MOMA.

Finally, we shall show that a new MOMA can also be used to build a CRT converter faster than recent design by Lee *et al.* [21], [22]. The new CRT converter differs from the one from [22] only in the implementation of the FC (see Fig. 2). Since r, the number of moduli which form an RNS, usually does not exceed eight and the actual number of residues mod A to be added can be reduced from r to k by forming partial sums (as suggested e.g. in [22]), we can assume that $k\leq 8$. Thus, we have $t_{Conv(b+1)}=3$ and therefore the delay of the parallel and word-serial converter using CLA equals $t_P(k, A)=2\theta(k)+29$ and $t_{WS}(k, A)=2k+27$, respectively. Table VII compares speed of the various architectures of the CRT converters with $a=32$ (input ROM's excluded). The new parallel CRT converter is faster than the word-serial design from [20] and the parallel design from [21]. The new word-serial converter is not only faster than one from [20], but is is also faster and less complex than the parallel converter from [21]. (Note: $k=10$ is the maximutm number of relatively prime moduli for $a=32$.)

TABLE VII
SPEED COMPARISON OF THE CRT CONVERTERS FOR SOME A WITH $a=32$

k	3	5	7	9
P	31	35	37	37
WS	33	37	41	45
[20]	34	54	74	94
[21]	40	45	47	47

VI. CONCLUSION

The new procedures for synthesizing residue generators and MOMA's using CSA are presented. First, the periodic properties of the series of powers of 2 taken mod A, previously observed and exploited for $A=2^a-1$, are extended to any odd A. Similarly, the well known equation $|2^{a-1}|_{2^{a-1}+1}=-1$ which holds for $A=2^{a-1}+1$ is extended to any odd A for which there exists $j>0$ such that $|2^j|_A=-1$. Then, due to periodicity, it is shown that a CSA/CPA network with EAC can be built for many A but not only for $A=2^a-1$ as it has been thought to date. Four new highly parallel schemes of the n-input generator mod A are proposed to suit various needs of A and n. The best performance offer the generators employing the CSA/CPA network with EAC (i.e., those with small $Per(A)$). The new generators are shown faster and less costly than conventional designs and, unlike other designs, for some A and n they can be pipelined on a full-adder level.

The concepts used to design residue generators are extended to design new (k, A) MOMA's (k is the number of operands). The design procedures for parallel as well as word-serial MOMA's are given. (A by-product of our approach is a

new efficient two-operand parallel adder mod $2^{a-1}+1$.) Comparison with existing MOMA's reveals significant speed improvement and in some cases also hardware reduction. The Chinese remainder theorem converter realized using a new MOMA introduces about 25% less delay than the fastest design reported in the literature.

We believe that the results presented here will be beneficial for researchers and practitioners working on hardware supporting highly-reliable digital systems protected against errors by using arithmetic codes and high-performance RNS-based DSP systems. The availability of efficient residue generators for any module A may originate interest for application of arithmetic error detecting codes with check bases other than $A = 2^a - 1$, since low-cost encoding and decoding circuitry can be constructed for them now. New pipelined residue generator used as an encoder or checking circuit is particularly well suited to support any arithmetic circuit pipelined at the FA level, which is protected against errors by an arithmetic code. It seems that the new periodicity measure $Per(A)$ defined in Section II has become an important criterium of selecting moduli A to form an RNS. One example is the three-moduli RNS $\{2^a-1, 2^a, 2^{a-1}+1\}$ and the other is the six-moduli RNS $\{A_i, 1 \le i \le 6\} = \{5, 7, 9, 11, 13, 16\}$ with $\max\{Per(A_i)\} = 6$ and $\max\{a_i\} = 4$, whose dynamic range $M = \prod_{i=1}^{6} A_i > 2^{19}$ is sufficient for many DSP applications. This is because for small $Per(A)$, the implementation cost of an n-input generator mod A and a parallel (k, A) MOMA is approximated by n (or $k \cdot a$) full-adders, respectively. The new (k, A) MOMA's are at their best for A with small $Per(A)$ and their speed advantage and hardware efficiency over known designs grows with k. They offer some hardware savings and speed improvement compared to analogous networks without EAC virtually at no cost and a word-serial MOMA with EAC has no architectural constraints on the number of operands k.

ACKNOWLEDGMENT

The author thanks the reviewers for their criticism and Artur Wrzyszcz for many suggested improvements and corrections.

REFERENCES

[1] A. Avižienis, "A set of algorithms for a diagnosable arithmetic unit," Tech. Rep. 32-546, Jet Propulsion Lab., Calif. Inst. Technol., Pasadena, CA, Mar. 1964.

[2] A. Avižienis, "Arithmetic codes: Cost and effectiveness studies for applications in digital system design," *IEEE Trans. Comput.*, vol. C-20, pp. 1322–1331, Nov. 1971.

[3] J. F. Wakerly, *Error Detecting Codes, Self-Checking Circuits and Applications.* New York: North-Holland, 1978.

[4] S. J. Piestrak, "Self-testing checkers for arithmetic codes with any check base A," in *Proc. 1991 Pacific Rim Int. Symp. Fault-Tolerant Syst.*, Kawasaki, Japan, Sept. 26–27, 1991, pp. 162–167.

[5] V. Piuri, M. Berzieri, A. Bisaschi, and A. Fabi, "Residue arithmetic for a fault-tolerant multiplier: The choice of the best triple of bases," *Microproc. and Microprogr.*, vol. 20, pp. 15–23, 1988.

[6] T. J. Slegel and R. J. Veracca, "Design and performance of the IBM Enterprise System/9000 Type 9121 vector facility," *IBM J. Res. Develop.*, vol. 35, pp. 367–381, May 1991.

[7] S. R. Barraclough *et al.*, "The design and implementation of the IMS A110 image and signal processor," in *Proc. IEEE Custom Integr. Circuits Conf.*, 1989, pp. 24.5.1–24.5.4.

[8] M. A. Soderstrand, "A new hardware implementation of modulo adders for reside number systems," in *Proc. 26th Midwest Symp. Circuits Systems,* 1983, pp. 412–415.

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

[10] M. A. Soderstrand *et al.*, Eds., *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing.* New York: IEEE Press, 1986.

[11] W. K. Jenkins and B. J. Leon, "The use of residue number system in the design of finite impulse response filters," *IEEE Trans. Circuits Syst.*, vol. CAS-24, pp. 191–201, Apr. 1977.

[12] M. Taheri, G. A. Jullien, and W. C. Miller, "High-speed signal processing using systolic arrays over finite rings," *IEEE J. Selected Areas Commun.*, vol. 6, pp. 504–512, Apr. 1988.

[13] C. N. Zhang, G. A. Jullien, and W. C. Miller, "Recursive reduction in finite ring computations," in *Proc. Conf. Rec. 23th Asilomar Conf. Circs., Systs, Comput.*, 1989, pp. 854–857.

[14] C. N. Zhang, G. A. Jullien, and W. C. Miller, "A neural-like network approach to finite ring computations," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 1048–1052, Aug. 1990.

[15] R. M. Capocelli and R. Giancarlo, "Efficient VLSI networks for converting an integer from binary system to residue number system and vice versa," *IEEE Trans. Circuits Syst.*, vol. CAS-35, pp. 1425–1430, Nov. 1988.

[16] G. Alia and E. Martinelli, "VLSI binary-residue converters for pipeline processing," *Comput. J.*, vol. 33, pp. 473–475, no. 5, 1990.

[17] K. Hwang, *Computer Arithmetic: Principle, Architecture ad Design.* New York: Wiley, 1979.

[18] C. N. Zhang, B. Shirazi, and D. Y.Y. Yun, "Parallel designs for Chinese remainder conversion," in *Proc. Int. Conf. Parallel Processing*, Aug. 17–21, 1987, pp. 557–559.

[19] L. Skavantzos, "Design of multi-operand carry-save adders for arithmetic modulo (2^n+1)," *Electron. Lett.*, vol. 25, no. 17, pp. 1152–1153, Aug. 17, 1989.

[20] C. K. Koç and C. Y. Hung, "Multi-operand modulo addition using carry save adders," *Electron. Lett.*, vol. 26, no. 6, pp. 361–363, Mar. 15, 1990.

[21] K. P. Lee, M. A. Bayoumi, and K. M. Elleithy, "A fast and flexible residue decoder based on the Chinese Remainder Theorem," in *Proc. ISCAS'89,* 1989, pp. 200–203.

[22] K. M. Elleithy, M. A. Bayoumi, and K. P. Lee, "$\theta(\log N)$ architectures for RNS arithmetic decoding," in *Proc. 9th Int. Symp. Comput. Arithm.*, Santa Monica, CA, Sept. 3–6, 1989, pp. 202–209.

[23] I. S. Reed *et al.*, "VLSI implementation of GSC architecture with a new ripple carry adder," in *Proc. ICCD'88,* 1988, pp. 520–523.

Stanislaw J. Piestrak was born on July 04, 1954 in Poland. He received the M.Sc. (with highest honors) and Ph.D. degrees, both in computer science, from the Technical University of Wrocław, Wrocław, Poland, in 1977 and 1982, respectively.

From 1977 to 1981 he was with the Institute of Engineering Cybernetics, Technical University of Wrocław. In the years 1982–1984 and 1987–1990, he was with the Institute of Power Systems Automation in Wrocław. Academic year 1984–1985, he spent as a Visiting Assistant Professor in the Center for Advanced Computer Studies, University of Southwestern Louisiana, Lafayette, LA. During two academic years 1985–1987 he was a Visiting Assistant Professor in the Computer Science Department, University of Georgia, Athens, GA. Since 1990, he has been with the Institute of Engineering Cybernetics, Technical University of Wrocław. From June to August 1993 he was on leave at TIMA/INPG Lab, Grenoble, France under a COST program. His research interests include design and analysis of hardware algorithms, fault-tolerant computing, self-checking circuits design, testing, coding theory, and computer arithmetic.