# Design of Residue Generators and Multi-Operand Modular Adders Using Carry-Save Adders

Stanisław J. Piestrak

Technical University of Wrocław
Institute of Engineering Cybernetics
50-370 Wrocław, POLAND
sjp@ict.pwr.pl

## Abstract

*In this paper, the design of residue generators and multi-operand modular adders is studied. New highly-parallel schemes using carry-save adders with end-around carry are proposed for either type of circuit. They are derived on the basis of the periodicity of the series of powers of 2 taken modulo A (A is a module). The new circuits are faster and use less hardware than similar circuits known to date.*

## 1 Introduction

Residue arithmetic has been used in digital computing systems for various purposes for many years, see for example [1], [2], [4], and [3]. The two most important applications of residue arithmetic are:

- high-performance digital signal processing hardware which makes use of the parallel nature of the residue number systems (RNSs) arithmetic; and
- fault-tolerant digital systems protected against undetected errors using arithmetic error detecting and/or error correcting codes.

Either class of a system uses a *generator mod A*, i.e. a circuit that computes $[X]_A$, the residue modulo $A$, from a given binary number $X$. In an RNS-based digital system it is used to build a binary-to-residue converter [1], [3], while in a fault-tolerant digital system using an arithmetic error detecting code (EDC) it is used to build an encoding/decoding circuitry [5], [9], and [10]. 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 the minimal amount of hardware.

Consider a generator mod $A$ with $n$ inputs $\{x_{n-1}, \ldots, x_1, x_0\}$ and $a$ outputs $\{s_{a-1}, \ldots, s_1, s_0\}$, where $a = \lceil log_2 A \rceil$. 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

$$[X]_A = [\sum_{j=0}^{n-1} 2^j x_j]_A. \quad (1)$$

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. A more efficient algorithm given in [1], is based on the expression

$$[X]_A = [\sum_{j=0}^{n-1} [2^j]_A x_j]_A. \quad (2)$$

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 $n$ residues can be performed by a $\lceil \log(n-1) \rceil$-level tree of $n-1$ two-operand binary adders mod $A$, a regular but costly and time-consuming scheme (by log$n$ we always denote logarithm of $n$ to the base 2). This approach was the basis for implementations of universal area-time efficient VLSI networks for converting an integer from binary system to residue number system [14]. The residue generator from [14] uses $\lceil n/logn \rceil$ Brent-Kung parallel adders which perform the addition of two residues mod $A$ in $O(\log n)$ parallel steps, each step performed in $O(\log \log A)$=$O(\log a)$ time. This method aimed for RNS-based systems, where $n$ is relatively small, seems inefficient (both in terms of cost and speed) to realize encoding and decoding circuitry for arithmetic EDCs, where it is not uncommon that $n \geq 32$. An alternative method to construct the generator by using look-up table implemented with ROMs was considered in [13].

The generator mod $A = 2^a - 1$ needs a special consideration since the residue $[X]_{2^a-1}$ can be obtained by mod $2^a - 1$ addition of $a$-bit bytes of $X$, due to the identity

$$[2^{ta}]_{2^a-1} = 1. \quad (3)$$

Thus the computation of $[X]_{2^a-1}$ can be accomplished with a $\lceil \log(b-1) \rceil$-level tree of $b-1$ adders mod $2^a - 1$, where $b = \lceil n/a \rceil$. This elegant and efficient generator was proposed by Avizienis [11], [12] to implement checking algorithms for arithmetic codes. Since the generators available for $A \neq 2^a - 1$ has been significantly more complex than for $A = 2^a - 1$, arithmetic

Research supported by the Ministry of Education of Poland, Grant DNS-T/15/070/90-2.

EDCs with the check base $A = 2^a - 1$ have been called *low – cost arithmetic codes* since then. This scheme has been recommended for implementation of encoding/decoding algorithms for low-cost arithmetic codes in all the literature known to the author, e.g. [2], [4], [5], [6], [7], [8], and [9]. Also, only low-cost arithmetic codes have been used in practical applications reported to date.

A *multi – operand modular adder* (MOMA) is another circuit frequently used in modular arithmetic applications. In particular, it can be used to build a residue generator but, as far as we know, this concept has not been considered explicitly in the literature. The design of multi-operand adders mod $A$ was considered in [15] (only for $A = 2^a - 1$), and most recently in [19], [16], [17] (only for $A = 2^{a-1}+1$), and [18]. The MOMA from [19] implemented by associative table look-up processing is generally more complex than any scheme using carry-save adder (CSA) network. The most efficient are the designs from [15]–for $A = 2^a - 1$, and from [18]–for any other $A$. Only the design from [15] uses a CSA with end-around carry (EAC).

In this paper, we show how to design new faster and less complex residue generators and MOMAs for any $A$. The design of either circuit is based on the extensive use of highly-parallel circuitry such as a CSA network with EAC. The design of a CSA with EAC proposed here was made possible by exploiting the periodicity of the series of powers of 2 taken mod $A$. Actually, the residue generators proposed here are the first ever designs which use CSAs.

## 2 Periodic Properties of $[2^j]_A$

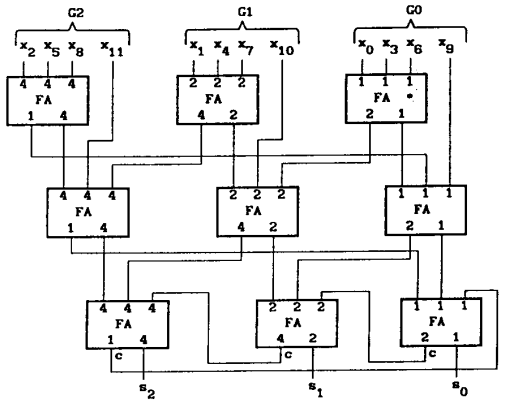
The periodicity of the series of powers of 2 taken mod $A$ is of key importance in the new design methods for the generator mod $A$ and multi-operand adder mod $A$ proposed in this paper. We start with the following definition adapted from [20].

*Definition* : 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 $A$. This is formally written as: $P(A) = \min\{j \mid j > 0$ and $[2^j]_A = 1\}$. Henceforth $P(A)$ will be simply called the *period of* $A$.

Table I. Periods $P(A)$ of odd modules $A \leq 65$.

| $A$ | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 |
|---|---|---|---|---|---|---|---|---|
| $P(A)$ | 2 | 4 | 3 | 6 | 10 | 12 | 4 | 8 |
| $A$ | 19 | 21 | 23 | 25 | 27 | 29 | 31 | 33 |
| $P(A)$ | 18 | 6 | 11 | 20 | 18 | 28 | 5 | 10 |
| $A$ | 35 | 37 | 39 | 41 | 43 | 45 | 47 | 49 |
| $P(A)$ | 12 | 36 | 12 | 20 | 14 | 12 | 23 | 42 |
| $A$ | 51 | 53 | 55 | 57 | 59 | 61 | 63 | 65 |
| $P(A)$ | 8 | 52 | 20 | 18 | 58 | 60 | 6 | 12 |

(A)




(B)

| $G2$ | $G1$ | $G0$ |
|---|---|---|
| 4 | 4 | 4 |
| ← 3-Bit CSA → | | |
| 3 | 3 | 3 |
| ← 3-Bit CSA → | | |
| 2 | 2 | 2 |
| ← 3-Bit Cyclic Adder → | | |
| 1 | 1 | 1 |
| $s_2$ | $s_1$ | $s_0$ |

Figure 1. CSA network with EAC for 4-operand adder mod 7 or 12-bit generator mod 7: (A) detailed scheme; and (B) shorthand description.

$P(A)$ can be calculated by using the following recursive equation

$$[2^i]_A = [\, 2[2^{i-1}]_A \,]_A. \tag{4}$$

However, for some $A$ the following formulas are known to compute $P(A)$ [20]:

- $P(2^a - 1) = a$;
- $P(2^{a-1} + 1) = 2(a - 1)$;
- If $A = kB$ with $k$ and $B$ odd then $P(A)$ is some multiple of $P(B)$.

Now, with the concept of the period of $A$, we generalize the identity (3) which holds for $A = 2^a - 1$ only, to the following identity which holds for any $A$

$$[2^{tP(A)}]_{2^{P(A)}-1} = 1, \tag{5}$$

where $t$ is any nonnegative integer.

## 3 Carry-Cave Adder Networks with End-Around Carry

The concept of *carry – save addition* has been used for many years to speed up multi-operand binary addition [15] and to implement MOMAs [15], [16], [17], and [18]. The MOMA for $A = 2^a - 1$ given in [15] (pp. 98-100) is the only example of using a *CSA with EAC* that has been reported in the literature. It is illustrated with the following example.

*Example* 1 : Design the 4-operand adder mod 7 using a CSA with EAC. The input 12-tuple are four residues mod 7: $X_1 = \{x_2x_1x_0\}$, $X_2 = \{x_5x_4x_3\}$, $X_3 = \{x_8x_7x_6\}$, and $X_4 = \{x_{11}x_{10}x_9\}$. The set of input bits is partitioned onto $a$=3 disjoint sets $Gj$ with the bits of the same weight $[2^j]_7$: $G0 = \{x_0, x_3, x_6, x_9\}$, $G1 = \{x_1, x_4, x_7, x_{10}\}$, and $G2 = \{x_2, x_5, x_8, x_{11}\}$. Figure 1(A) shows a detailed scheme of the circuit, while Fig. 1(B) shows its shorthand description, where the columns $Gj$ indicate the number of bits of weight $[2^j]_7$ in the current stage of computation. The latter scheme will be used henceforth to describe any CSA network).

The operation of the above circuit as well as any other $a$-bit CSA network with EAC is justified by (3) for $A = 2^a - 1$. Since $P(2^a - 1) = a$, one can expect that a similar concept can be applied to any other $A$, provided that a $P(A)$-bit CSA network with EAC is considered and a more general identity (5) is taken into account. Below we will show that this is indeed the case.

Suppose that we wish to add $k \geq 3$ $P(A)$-bit numbers mod $2^{P(A)} - 1$, $A$ is an odd integer. A $P(A)$ – *bit CSA with EAC* is a natural generalization of the well known $a$-bit CSA with EAC used for $A = 2^a - 1$. This is because a FA with inputs of weight $[2^{P(A)-1}]_A$ generates the carry signal of weight $[2[2^{P(A)-1}]_A]_A = 1$ that can be directed to the FA with inputs of weight 1. Similarly, the $a$-bit binary adder with EAC used for $A = 2^a - 1$ can be generalized to a *P(A)-bit binary adder with EAC* used for any $A$. A remarkable feature of this circuit, which is of our particular concern, is the cyclic nature of the addition mod $A$ with the cycle length $P(A)$. It allows any $P(A)$-bit adder with EAC to begin and end operation at *any* point of the cycle. In our applications, this adder executes at most one full cycle of length $P(A)$. Thus, in the worst case, if the $P(A)$-bit adder with EAC begins operation from $Gj$, its operation ends at $G(j-1)$ with the last carry signal directed to $Gj$ (see, for example, Fig. 5). Henceforth, the $P(A)$-bit adder with EAC which goes through $p$ ($p \leq P(A)$) stages will be called a $p$ – *bit cyclic adder*. One limitation of the $P(A)$-bit adder is that its delay grows with $P(A)$ and it can be prohibitively large for some $A$. In such a case, we suggest to use the following scheme for large $P(A)$. Some other modifications of the new basic generator scheme, which allow to avoid this problem, will be given in the next section.

If $P(A)$ is large we can partition $P(A)$ cyclically ordered sets $Gj$ onto $w$ groups $S_i$, each group $S_i$ containing *subsequent* sets $Gj$, $1 \leq i \leq w$. If the groups $S_i$ are of similar sizes, the addition of two $P(A)$-bit operands can be performed in parallel by $w$ shorter binary adders of length no more than $\lceil P(A)/w \rceil$. This implementation involves $w - 1$ extra inputs to the final converter of the generator (see Fig. 3 and Section 4), since now there are $w > 1$ sets $Gj$ with two bits. The following example clarifies such a case.

*Example* 2 : A CSA network with EAC for the 32-bit generator mod 13 is given on Fig. 2. The set of 32 input bits is partitioned onto $P(13) = 12$ sets $Gj$. The 8-bit CSA reduces the input bits from $n = 32$ to $n' = 24$,

| G11 | G10 | G9 | G8 | G7 | G6 | G5 | G4 | G3 | G2 | G1 | G0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| HA | HA | HA | HA | ← 8-Bit CSA → | | | | | | | |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| ← 6-Bit Adder → | | | | | | ← 6-Bit Adder → | | | | | |
| 1 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 2 |

Figure 2. CSA network for 32-input generator mod 13.

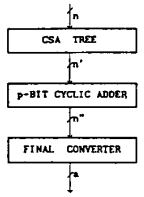



Figure 3. General structure of the new $n$-input generator mod $A$ and the $k$-operand adder mod $A$.

which are uniformly distributed due to four HAs. Now a 12-bit adder with EAC could be used to reduce the bits from $n' = 24$ to $n" = 13$. However, if it is too slow a viable alternative are two 6-bit adders used as shown on Fig. 2.

## 4 New Generators Mod A

In this section, we propose three new schemes of the $n$-input generator mod $A$ which cover the whole spectrum of $A$ and n; two of them make use of the CSA network with EAC.

### 4.1 General Design Procedure

Here we will propose a new $n$-input generator mod $A$ that computes (2) using a CSA network.

Let $n = uP(A) + v$, $u$=0,1,2,... and $0 \leq v \leq P(A) - 1$. For $n > P(A)$ we have $u \geq 1$ and $v > 0$. It follows from (5) that for any $j$ ($0 \leq j \leq P(A) - 1$) and any nonnegative integer $t$

$$[2^{t(PA)+j}]_A = [2^j]_A, \tag{6}$$

which implies that $u + 1$ bits $x_j, x_{P(A)+j}, \ldots, x_{uP(A)+j}$ represent the same residue $[2^j]_A$ for any $j \leq v$. The same holds for $u$ bits $x_j, x_{P(A)+j}, \ldots, x_{(u-1)P(A)+j}$ for any $j > v$. This leads us to the partitioning of $n$ input bits onto $P(A)$ subsets $Gj$ defined as $Gj = \{x_q \mid ([2^q]_A = j) \text{ and } (0 \leq q \leq n-1)\}$, $j = \{0, 1, 2, \ldots, P(A) - 1\}$. The sets $Gj$ are cyclically ordered according to increasing indices $j$ taken mod $P(A)$; in particular, the sets $G(P(A)-1)$ and $G0$ are adjacent and hence $G0$ follows $G(P(A) - 1)$. This leads us to a new general structure of the $n$-input generator mod $A$, shown on Fig. 3. It is designed by using the following procedure.

Procedure 1

**Step 1:** Partition the set of input bits $\{x_0, x_1, \ldots, x_{n-1}\}$ onto $P(A)$ sets $Gj = \{x_q \mid [2^q]_A = j\}$, $0 \leq j \leq P(A) - 1$.

**Step 2:** If $n \leq 2P(A) + 1$ assume $n' = n$ and go to Step 3. Otherwise, reduce the total number of bits from $n$ to $n'$ ($n' = 2P(A)$ or $2P(A) + 1$) using a

CSA with EAC of length up to $P(A)$.
For $n' = 2P(A)$ any set $Gj$ has two bits, whereas for $n' = 2P(A)+1$ one set, say $Gj^*$, has three bits while all other sets $Gj$ have two bits.

**Step 3:** Reduce the number of bits from $n'$ to $n" = P(A)+1$ by using a $p$-bit cyclic adder.

**a)** If $n' = n$ then $p = n' - P(A)$ and the adder starts with $G0$;

**b)** If $n' = 2P(A)$ then $p = P(A)$ and the adder can start with any set $Gj$;

**c)** If $n' = 2P(A)+1$ then $p = P(A)$ and the adder starts with $Gj^*$.

In any case, every set $Gj$ but one, say $Gr$, contains a single bit at the end of the addition; depending on the case $r$ equals: a) $n' - P(A)$; b) $j+1$; or c) $j^*+1$.

**Step 4:** Compute $[\, y_0[2^0]_A + y_1[2^1]_A + \ldots + y'_r[2^r]_A + y''_r[2^r]_A + \ldots + y_{P(A)-1}[2^{P(A)-1}]_A \,]_A$.

Step 4 is executed by the final converter which is nothing else but an $n"$-input generator mod $A$. It can be implemented using one of the schemes proposed in Subsections 4.2 and 4.3. The final converter is not used for $A = 2^a - 1$ because $n' = a$.

Basically, Proc. 1 can be used to design a generator for any $A$ and $n$ such that $n > P(A)$, but its efficiency grows with the ratio $n/P(A)$. The major limitation of Procedure 1 is that we cannot take the full advantage of an extensive use of the CSA network (if at all) when $n/P(A) < 3$, and in such a case most bits must be reduced by the final converter. Table I shows that there are many $A$ for which $P(A)$ is small, e.g. for $A \in \{5, 9, 17, 21, 51\}$ we have $P(A) \leq 8$, for which an efficient generator can be designed even for small $n$. Unfortunately, $P(A)$ can also be prohibitively large, e.g. for $A \in \{29, 37, 49, 53, 59, 61\}$ we have $P(A) \geq 28$, and for any such $A$, Procedure 1 can provide an efficient generator for very large $n$ only. Thus, for any pair of $n$ and $A$ with a small ratio $n/P(A)$ two special schemes of the generator mod $A$ are proposed below. They can be used to implement the whole $n$-input generator or only the $n"$-input final converter for the generator designed by using Proc. 1.

*Example* 3 : A CSA network for the 32-bit generator mod 9 is given on Fig. 4. The input bits are partitioned onto $P(9) = 6$ sets $G0 = \{x_0, x_6, x_{12}, x_{18}, x_{24}, x_{30}\}$, ..., and $G5 = \{x_5, x_{11}, x_{17}, x_{23}, x_{29}\}$. The whole CSA network is built of 25 FAs and one HA and it introduces the delay of $9\Delta$ ($\Delta$ is the delay introduced by a single FA). The final converter which computes

$$[\, 1.y_0 + 2.y_1 + 4.y'_2 + 4.y''_2 + 8.y_3 + 7.y_4 + 5.y_5 \,]_A.$$

can best be realized with a $128 \times 4$ ROM.

## 4.2 Generator Mod A for n≤P(A)

If $n \leq P(A)$ then every coefficient $[2^i]_A$, $0 \leq i \leq n-1$, represents a different residue mod $A$. Thus, to compute (2) we propose a special scheme, shown on Fig. 5. The set of $n$ input bits is partitioned onto two

| $G5$ | $G4$ | $G3$ | $G2$ | $G1$ | $G0$ |
|---|---|---|---|---|---|
| 5 | 5 | 5 | 5 | 6 | 6 |
| ← 6-Bit CSA → | | | | ← 2-Bit CSA → | |
| 4 | 4 | 4 | 5 | 4 | 3 |
| ← 6-Bit CSA → | | | | | |
| 3 | 3 | 3 | 4 | 3 | 2 |
| ← 5-Bit CSA → | | | | | HA |
| 2 | 2 | 2 | 3 | 2 | 2 |
| 6-Bit Cyclic Adder | | | | | |
| 1 | 1 | 1 | 2 | 1 | 1 |
| $y_5$ | $y_4$ | $y_3$ | $y''_2, y'_2$ | $y_1$ | $y_0$ |

Figure 4. CSA network for 32-bit generator mod 9.

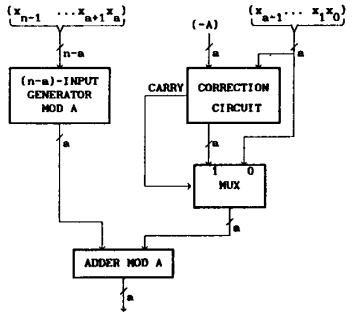



Figure 5. New generator mod $A$ for $n \leq P(A)$.

disjoint subsets $X_1 = \{x_{n-1}, \ldots, x_{a+1}, x_a\}$ and $X_2 = \{x_{a-1}, \ldots, x_1, x_0\}$. The $n-a$ bits from $X_1$ feed the $(n-a)$-input generator mod $A$ that computes

$$[\, x_a[2^a]_A + x_{a+1}[2^{a+1}]_A + \ldots + x_{n-1}[2^{n-1}]_A \,]_A \quad (7)$$

and should be implemented with a PLA or ROM. The remaining $a$-tuple $X_2 = (x_{a-1} \ldots x_1 x_0)$ represents an integer $M$, $0 \leq M \leq 2^a - 1$, which can be treated as a residue mod $A$ with or without overflow. To find $[M]_A$ a correction circuit with a multiplexer is employed. The correction circuit that computes $M - A$ can be implemented either as an $a$-bit adder with a PLA or as a ROM correction table. The carry bit generated from the correction circuit indicates whether $M$ is greater than $A$. A multiplexer controlled by the carry selects the correct output from $M$ and $M - A$.

The range of applications of the above scheme is limited, probably to $10 < n \leq 10 + a$, because of the following reasons. For $n \leq 10$ the whole generator can best be implemented with a single specially tailored $2^n \times a$ ROM. For $n > 10 + a$ the $(n-a)$-bit generator mod $A$ would require more than one ROM and then a pure look-up table or any other implementation of the whole $n$-bit generator could be better. Nevertheless, this scheme is more efficient than any scheme based on the concept from [1], since, if implemented using modular adders, it uses $a$-1 modular additions less.

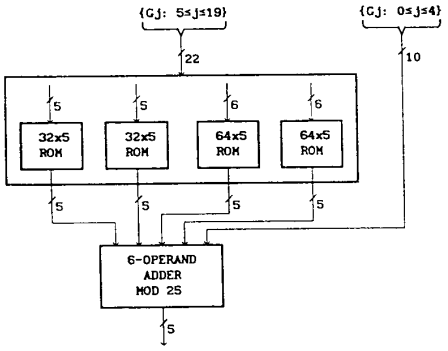



Figure 6. 32-input generator mod 25 using 6-operand adder mod 25.

### 4.3 Generator Mod A Using a MOMA

Here we propose the third scheme of the residue generator which handles the whole range of $A$ and $n$ for which two schemes proposed above are inefficient. The scheme described here first translates the groups of those bits which cannot be reduced by using a CSA network onto residues mod $A$. These residues are then added mod $A$ by a MOMA. The efficiency of this scheme is a natural consequence of using a CSA-based MOMA (described in Section 5).

*Example* 4 : Design the 32-input generator mod 25. First consider the generator designed by using Proc. 1. The input bits are partitioned onto $P(25) = 20$ sets $Gj = \{x_j, x_{j+20} \mid 0 \leq j \leq 11\}$ and $Gj = \{x_j \mid 12 \leq j \leq 31\}$. Thus the whole generator consists of a 12-bit binary adder with inputs from the sets $Gj$ with $0 \leq j \leq 11$ and the final converter with 21 inputs. A ROM implementation of the latter needs one $2K \times 5$ ROM and one $1K \times 5$ ROM at the first stage, and one $2K \times 5$ ROM at the second stage.

A significantly more efficient generator, shown on Fig. 6, uses the 6-operand adder mod 25. Ten bits from the sets $Gj$, $0 \leq j \leq 4$, are treated as two operands (the fact that they may not be residues mod 25 is immaterial here), while remaining four operands are residues mod 25 generated by four small ROMs (or PLAs) from 22 bits from the sets $Gj$, $5 \leq j \leq 19$. The 6-operand adder mod 25 can be designed by using Proc. 2 given in Section 5: it is built of 23 FAs, one HA, and one $256 \times 5$ ROM, and it introduces the delay of $7\Delta + d(\text{ROM})$.

### 4.4 Complexity Estimation

The CSA network that reduces the number of bits from $n$ to $n"$ is built of a total of $n - n"$ FAs and a few HAs. Hardware complexity of the final converter depends on a particular scheme used. The delay $d(A, n)$ introduced by the generator designed by using Proc. 1 equals

$$d(A, n) = \theta(k).\Delta + d(p) + d(A, n"), \qquad (8)$$

where:

- $\theta(k)$, the minimum number of levels in the CSA tree with $k$ operands ($k$, can be found in Table II, where: $k = u$—for $v$={0 or 1},and $k = u + 1$—for $v > 1$.

Table II. The minimum number of levels $\theta(k)$ on a CSA tree that processes $k$ input operands [15].

| k | 3 | 4 | 5÷6 | 7÷9 | 10÷13 | 14÷19 | 20÷28 |
|---|---|---|---|---|---|---|---|
| $\theta(k)$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

- $d(p)$ is the delay introduced by the $p$-bit binary adder; and
- $d(A, n")$ is the delay introduced by the final converter.

It is clear that the delay of the CSA tree is significantly smaller than the delay introduced by the $p$-bit adder and the final converter. Therefore, to improve the speed of the generator, it can be desirable to implement either the final converter or even the $p$-bit adder and the final generator together, with a $2^{n"} \times a$ or $2^{n'} \times a$ ROM, respectively. On the other hand, the hardware complexity of the generator can be reduced in various ways at the cost of speed. For instance, for $k > 4$ a viable alternative is to replace the CSA tree with a single bank of $P(A)$ FAs followed by a $2P(A)$-bit carry-save register (CSR) which operates in $k - 2 + p$ cycles. The trade-off is a $2P(A)$-bit CSR and extra delay of $[k - 2 - \theta(k)]\Delta$ for $n - n' - P(A)$ FAs less.

### 4.5 Comparison

The superiority of the designs proposed here over conventional approaches results from the following argument.

Any new generator operates as the circuit that reduces the number of input bits with weights being residues mod $A$ from $n$ to $a$. Thus, it seems appropriate to estimate the average amount of hardware required to reduce the number of bits by one. With a CSA network, one FA reduces the number of bits by one. To the author's best knowledge, there is no other scheme that does the same with less hardware and no residue generator using a CSA network has been reported in the open literature as well. For comparison, any generator based on the concept from [1] uses two $a$-bit adders to reduce the number of bits by one while a look-up table approach from [13] is more complex as well. These observations support our claim that any generator proposed here uses less hardware than any other existing design. The more so that the schemes used to implement the final converter, proposed in Subsections 4.2 and 4.3, are also less complex than existing designs.

The main contributors to the delay introduced by the generators proposed here are the $p$-bit cyclic adder and the final converter. Since the delay of the $p$-bit cyclic adder can be as large as $P(A)$, its partitioning onto smaller adders (suggested in Section 3) or implementing the whole generator mod $A$ (or the final converter only) using a MOMA, can be used to improve performance of a generator. Thus, any new generator can be implemented using at most two relatively short binary adders. This clearly shows its superior speed compared to any scheme based on the concept from [1] which introduces the delay equal $2\lceil log(n-1)\rceil.d(a).\Delta$.

Finally, we should comment a new scheme of the generator for a special case of $A = 2^a - 1$ which is very similar to the $k$-operand adder mod $A = 2^a - 1$ from [15]. The only difference between them is that the generator may have $n \neq ka$ inputs and needs $\lceil n/a \rceil a - n$ extra HAs for better performance. This now obvious and simple generator should be confronted with the scheme that has been commonly used to build the error checking circuitry for arithmetic codes with $A = 2^a - 1$ for about thirty years [11]. It has benn a tree of $a$-bit adders with EAC, despite that a significantly faster generator could have been built using CSA approach. This claim is true for all monographs on the design of fault-tolerant hardware known to the author (including [4], [5], [6], [7], and [8]) as well as for all papers on self-testing checkers for arithmetic codes with $A = 2^a - 1$, see e.g. [9]. The superiority of the CSA-based generator mod $2^a - 1$ over the conventional scheme is clear: either scheme is built of $n - a$ FAs, but the new one introduces a significantly smaller delay — $[\theta(\lceil n/a \rceil)+2a]\Delta$ *vs.* $2a\lceil \log(n/a) \rceil \Delta$. (It is assumed that the delay introduced by the $a$-bit adder with EAC is $2a\Delta$.)

## 5 Multi-Operand Adder Mod A

Let $X_1 = (x_{a-1}, \ldots, x_1, x_0)$, $X_2 = (x_{2a-1}, \ldots, x_{a+1}, x_a)$, $\ldots$, and $X_k = (x_{ka-1}, \ldots, x_{(k-1)a})$ be $k$ residues mod $A$. A $k$-operand adder mod $A$ computes

$$[X]_A = [\ \sum_{i=0}^{k} X_i\ ]_A \tag{9}$$

which is equivalent to

$$[X]_A = [\ \sum_{j=0}^{a} (\sum_{i=0}^{k} x_{ia+j}) \cdot [2^j]_A\ ]_A \tag{10}$$

The new technique proposed here first performs the summation

$$[\ \sum_{j=0}^{a} (\sum_{i=0}^{k} x_{ia+j}) \cdot [2^j]_A\ ]_{2^{P(A)}-1} \tag{11}$$

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

$$[\ [\ \sum_{j=0}^{a} (\sum_{i=0}^{k} x_{ia+j}) \cdot [2^j]_A\ ]_{2^{P(A)}-1}]_A \tag{12}$$

using a look-up table.

### 5.1 Design Procedure

A new $k$-operand adder mod $A$ that implements (12) employs similar ideas and the same general structure as the generator mod $A$, shown on Fig. 3.

The maximum of sets $Gj$ which will eventually be used in the CSA network of the new MOMA is given by

$$q = \min\{P(A), m\}. \tag{13}$$

where $m = \lceil \log k(A-1) \rceil$ is the number of bits needed to encode the largest number resulting from the summation of $k$ residues mod $A$ in binary.
A new $k$-operand adder mod $A$ can be designed by using the following procedure.

Procedure 2

**Step 1:** Find $m = \lceil \log k(A-1) \rceil$.

| $G3$ | $G2$ | $G1$ | $G0$ |
|---|---|---|---|
| — | 4 | 4 | 4 |
| — | ← 3-Bit CSA → | | |
| 1 | 3 | 3 | 2 |
| — | ←2-Bit CSA→ | | — |
| 2 | 2 | 1 | 2 |
| ┌FA←— | HA | ←——— | FA←┐ |
| 1 | 1 | 2 | 1 |

Figure 7. CSA network for 4-operand adder mod 5.

| G7 | G6 | G5 | G4 | G3 | G2 | G1 | G0 |
|---|---|---|---|---|---|---|---|
| — | — | — | 8 | 8 | 8 | 8 | 8 |
| — | — | — | ←—Two 5-Bit CSAs—→ | | | | |
| — | — | 2 | 6 | 6 | 6 | 6 | 4 |
| — | — | — | ← —5-Bit CSA—→ | | | | |
| — | — | — | ←—4-Bit CSA—→ | | | | |
| — | — | 4 | 4 | 4 | 4 | 3 | 2 |
| — | — | ←—5-Bit CSA—→ | | | | | HA |
| — | 1 | 3 | 3 | 3 | 3 | 2 | 1 |
| — | — | ←—4-Bit CSA—→ | | | | HA | — |
| — | 2 | 2 | 2 | 2 | 2 | 1 | 1 |
| — | ←—5-Bit Adder—→ | | | | | — | — |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Figure 8. CSA network for 8-operand adder mod 25.

**Step 2:** Let $\{X_i = (x_{ia-1}, \ldots x_{(i-1)a}),\ 1 \leq i \leq k\}$ be the set of input operands. Partition $n = ka$ input bits onto $a$ subsets $Gj = \{x_{ra+j} \mid 0 \leq r \leq k-1\}$, $0 \leq j \leq a-1$, and assume $Gj = \emptyset$, $a \leq j \leq m$.

**Step 3:** Reduce the total number of bits from $n$ to $n'$ by using the CSAs with EAC of length up to $q$ ($q = \min\{P(A), m\}$) until at most one set $Gj$ has three bits while all other sets $Gj$ have two or one bit.

**Step 4:** Reduce the number of bits from $n'$ to $n$" by using a $p$-bit cyclic adder (usually $n" = q$ or $q+1$).

**Step 5:** Compute $[\ y_0[2^0]_A + y_1[2^1]_A + \ldots + y_r'[2^r]_A + y_r"[2^r]_A + \ldots + y_{m-1}[2^{m-1}]_A\ ]_A$. by using one of the realizations of the final converter described in Section 4; $Gr = \{y_r, y_r"\}$ denotes the set with two bits, provided that such a set occurs.

### 5.2 Examples

The following examples illustrate the operation with and without cyclic mode of the CSA network used in the MOMA designed according to Proc. 2.

*Example* 5 : A CSA network for the 4-operand adder mod 5 is shown on Fig. 7. It uses seven FAs and one HA, and its delay is $5\Delta$. The final converter can best be implemented with a 5-input PLA. An alternative implementation of the CSA network using a single-stage 4-bit CSA and an 8-bit CSR introduces the same delay. It uses three FAs less at the cost of an 8-bit CSR.

*Example* 6 : A CSA network for the 8-operand adder mod 25 is given on Fig. 8. Since $P(25) = 20$ and $m = \lceil \log 192 \rceil = 8$ we have $q$=min$P(25), m$=8 and the cycle does not occur. Its high-speed (HS) version uses 32 FAs, two HAs, and 256×5 ROM, and introduces the delay of $7\Delta + d$(ROM). Its cost-effective (CE) version uses seven FAs, a 14-bit CSR, and 256×5 ROM, and introduces the delay of $11\Delta + d$(ROM). In this case, the CE version is a viable alternative to the HS version since the former is slightly slower (by $2\Delta$) but allows to save a large amount of hardware.

Notes:

1. The analysis of many examples showed that if HAs are used then the $p$-bit cyclic adder has $p \leq a$.
2. For large $k$, a more viable is to implement the CSA network using a single-stage $q$-bit CSA with a $2q$-bit CSR ($q \leq m+1$) which operates in $k-2+a$ or $k-3+a$ cycles.

## 5.3 Complexity Estimation of the New MOMA

The complexity of two basic blocks of the new MOMA: the CSA network and the final converter, will be considered separately.

The CSA network used in the new MOMA reduces $n = ka$ input bits to $n$", i.e. it is built of $ka - n$" FAs. Most of the FAs work in the carry-save mode, since the length of a single cyclic adder is $a$ or $a$-1 only. Thus the delay of the CSA network is upper-bounded by $[\theta(k) + a]\Delta$.

Now recall that the final converter is a special case of the generator mod $A$, in which the CSA network can hardly be used. Since its ROM (or PLA) implementation seems the most feasible, its complexity can be appropriately estimated by the number of inputs $n$". Obviously, $n$" is upper-bounded by $P(A) + 1$ for any $k$. An equality holds when the CSA network operates in the cyclic mode, i.e. for any $k \geq k_c(A)$, where

$$k_c(A) = \lceil 2^{P(A)}/(A-1) \rceil. \qquad (14)$$

For $A$ odd we distinguish the following cases:

**(a)** for $A = 2^a - 1$ the cyclic mode occurs for any $k$;

**(b)** for $A$ with $P(A) \leq 8$ the values of $k_c(A)$ are tabulated in Table III;

**(c)** for any other $A$ we have $P(A) \geq 10$ and $k_c(A) \geq 32$. Thus, it seems unlikely that for any such $A$ the cyclic mode occurs for any practical $k$.

Since in the case (b) we have $n$" $\leq 9$ for any $k$, the implementation of the final converter with a single ROM is feasible.

Now we will consider the case (c). Suppose that $m \leq P(A)$. It implies that either $n$" $= m$ or $n$" $= m+1$. The largest number of operands (which are residues mod $A$) such that their sum can be encoded with $m$ bits equals

$$k_m(A) = \lfloor (2^m - 1)/(A-1) \rfloor. \qquad (15)$$

The values of $k_8(A)$ for some $A$ are tabulated in Table IV. It is observed that for sufficiently large $a$ we have $k_m(2^{a-1}+1) = 2^{m-a+1} - 1$ and $k_m(2^a - 3) = 2^{m-a}$.

Table III. The values of $k_c(A)$ for $A$ with $P(A) \leq 8$.

| $A$ | 5 | 9 | 17 | 21 | 51 |
|---|---|---|---|---|---|
| $P(A)$ | 4 | 6 | 8 | 6 | 8 |
| $k_c(A)$ | 4 | 8 | 16 | 4 | 6 |

Table IV. The values of $k_8(A)$ for some $A$.

| $a$ | 4 | | 5 | | | | 6 | |
|---|---|---|---|---|---|---|---|---|
| $A$ | 11 | 13 | 17 | 19 | 23 | 29 | 33 | 61 |
| $k_8(A)$ | 23 | 19 | 15 | 13 | 11 | 8 | 7 | 4 |

Table V. Characteristics of various MOMAs.

| | Version | FAs | CSR | ROM | Delay |
|---|---|---|---|---|---|
| New | CE | $m$ | $2m$ | $\leq 2^{m+1}$ | $[\theta(k)+a]\Delta + d(ROM)$ |
| [18] | Basic | $2a+7$ | $4a+10$ | — | $(4k+2a-4)\Delta$ |
| New | HS | $\leq (k-1)a - m - 1$ | — | $\leq 2^{m+1}$ | $(k+a-2)\Delta + d(ROM)$ |
| [18] | Parallel | $3a+9$ | $4a+10$ | — | $(4k+a-6)\Delta$ |

*Note:* for $k \leq 16$ we have $m$=min$\{P(A), a+4\}$ and $\theta(k) \leq 6$.

(Note that $A = 2^{a-1}+1$ and $A = 2^a - 3$ are the smallest and the largest $A$ that can be encoded with $a$ bits in the case (c).) Table IV shows that for $m = 8$ these limits are attained for $a = 5$. It is clear that the final converter has no more than $a+4$ inputs for any $A$ and $k$ ranging from 16 up to 31. This proves that also in the case (c) the ROM implementation of the final converter is feasible for large $k$ and for many $A$, since for any $a \leq 6$ the ROM size does not exceed $1K \times a$.

## 5.4 Comparison

The MOMA from [15] is the most efficent scheme known for $A = 2^a - 1$; it is also the special case of Proc. 2. Therefore new MOMAs designed by Proc. 2 will be compared only for $A \neq 2^a - 1$ against the best known design obtained by using Algorithm C from [18].

Algorithm C from [18] computes (10) in the following way. During first $k$-2 cycles Step 2 of Alg. C is executed. Each cycle involves adding $X_i$ and subtracting $A$ performed by the $(a+3)$-bit CSA which are followed by the sign estimation of the partial sums, performed by the 2-bit carry-lookahead adder; the partial sums are stored in two $2(a+3)$-bit CSRs. The final reduction involves: either $2(a+2)$ additions performed by the $(a+2)$-bit CSA — in a basic version; or $a+2$ additions performed by two $(a+2)$-bit CSAs — in a parallel version. Either version uses two $(a+2)$-bit CSRs to store the results before the correct sum is chosen.

Table V compares the exact characteristics of the designs from [18] against the upper-bounds of the designs obtained by using Proc. 2. The comparison reveals that either version of our MOMA introduces significantly less delay than the design from [18]. Essential speed improvement can be attributed to the concept used in our design: the estimation of the sum is postponed to the last stage and, unlike in Alg. C from [18], no time is wasted for corrections and sign estimation during the carry-save stage of operation (they cause that the delay is proportional to $4k$ in Alg. C). Table V also reveals that our high-speed version becomes too complex when $k$ grows (probably for $k > 5$). However, the complexity

of our CE version and the basic version from [18] seem comparable.

## 6 Conclusions

The new procedures for synthesizing residue generators and multi-operand modular adders (MOMAs) using CSA are presented. First, it is observed that the periodic properties of the series of powers of 2 taken mod A, previously observed and exploited in [11] for $A = 2^a - 1$, can be extended to any $A$. It is shown that a CSA with EAC can be built for any $A$ but not only for $A = 2^a - 1$ as it has been thought to date. Based on the CSA with EAC network, the first ever $n$-input generator mod $A$ using CSA network is obtained. Three design schemes of the $n$-input generator mod $A$ are proposed to suit various needs for $A$ and $n$. Any new generator is superior to any conventional design with respect to speed and amount of hardware used. In particular, it is shown that the CSA-based generator mod $A = 2^a - 1$ should replace a scheme commonly used in the literature (built of two-operand adders mod $A = 2^a - 1$), since it offers essential speed improvement virtually at no cost. Finally, a new general procedure of a MOMA using a CSA with EAC is proposed. For any $A \neq 2^a - 1$ the new scheme shows significant speed improvement compared to the existing designs. For many $A$ and $k$ ($k$ is the number of operands) the hardware reduction is also observed.

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 as well as high-performance RNS-based systems. In particular, they may originate interest for application of arithmetic EDCs with check bases other than $A = 2^a - 1$ since low-cost encoding and decoding circuitry can be constructed for them now. Also, these results may give a new insight into the selection criteria of moduli used to form an RNS.

## References


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

[2] F. F. Sellers, Jr., M.-Y. Hsiao, and L. W. Bearnson, *Error Detecting Logic for Digital Computers,* McGraw-Hill, New York, 1968.

[3] F. J. Taylor, "Residue arithmetic: A tutorial with examples," *Computer*, pp. 50-62, May 1984.

[4] T. R. N. Rao, *Error Coding for Arithmetic Processors,* Academic Press, New York, 1974.

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

[6] G. D. Kraft and W. N. Toy, *Microprogrammed Control and Reliable Design of Small Computers,* Prentice-Hall, Englewood Cliffs, N.J., 1981.

[7] P. K. Lala, *Fault-Tolerant and Fault-Testable Hardware Design,* Prentice-Hall, Englewood Cliffs, N.J., 1985.

[8] Y. Tohma, "Coding Techniques in Fault-Tolerant, Self-Checking, and Fail-Safe Circuits," Ch. 5 in *Fault Tolerant Computing: Theory and Techniques, Vol. I,* D. K. Pradhan, Ed., Prentice-Hall, Englewood Cliffs, N. J., 1986.

[9] D. Nikolos, A. M. Paschalis, and G. Philokyprou, "Efficient design of totally self-checking checkers for all low-cost arithmetic codes," *IEEE Trans. Comput.*, vol. C-37, pp. 807-814, July 1988.

[10] S. J. Piestrak, "Design of high-speed and cost-effective self-testing checkers for low-cost arithmetic codes," *IEEE Trans. Comput.*, vol. C-39, pp. 360-374, March 1990.

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

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

[13] 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.

[14] 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.

[15] K. Hwang, *Computer Arithmetic: Principles, Architecture and Design,* Wiley, New York, 1979.

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

[17] L. Skavantzos, "Design of multioperand carry-save adders for arithmetic modulo $(2^n + 1)$," *Electron. Lett.*, vol. 25, No. 17, pp. 1152-1153, 17th Aug. 1989.

[18] C. K. Koc and C. Y. Hung, "Multi-operand modulo addition using carry save adders," *Electron. Lett.*, vol. 26, No. 6, pp. 361-363, 15th Mar. 1990.

[19] C. A. Papachristou, "Associative table lookup processing for multioperand residue arithmetic," *JACM*, vol. 34, pp. 376-396, Apr. 1987.

[20] 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.