The CSI Journal on Computer Science & Engineering

The CSI Journal on
Computer Science and Engineering
Vol. 6, No. 2 & 4 (b), 2008
Pages 29-35
Short Paper

A Generic Modulo 2^n–1 Adder Based on Stored Negabit Representation of Residues

Ghassem Jaberipur

Department of Electrical and Computer Engineering, Shahid Beheshti University, Tehran, Iran

Abstract

Abstract studies in computer arithmetic, often lead to generic frameworks for concrete implementations. Computer arithmetic algorithms show variant performance potentials on alternative input and output encodings. Standard computer arithmetic components (e.g., full/half adder cells, carry look-ahead blocks) are once in a while enhanced in performance. Therefore any computer arithmetic unit (e.g., *n*-bit adders and multipliers, carry-save adders (CSA), etc), generic in design, can be implemented by replacing its generic parts by the best available, standard and functionally equivalent, components meeting specific design goals in time, area and power. Up-grading is easily possible via using new technologic advances.

In this paper, we study abstract algorithms and generic implementations for modulo 2^n–1 addition. The latter is popular in many application domains of residue number systems (e.g., digital signal processing) and other applications such as fault tolerant computing. We propose a novel representation for modulo 2^n–1 residues. This is composed of an unsigned *n*-bit number and a stored negabit as a second least significant bit (*lsb*). A supporting algorithm, computing *sum*+1, stores a (–1)-valued negabit as the second *lsb* of the sum in case of no carry-out. This simple trick allows for a generic delay-optimized modulo 2^n–1 adder composed of an *n*-bit CSA and a generic *n*-bit adder. In concrete implementations, each designer can use the best available CSA and *n*-bit adder that meet the prescribed design goals. Given that CSAs show less switching activity than *n*-bit adders, savings in power dissipation is expected in comparison with the best previous generic design which uses two *n*-bit adders

Keywords: Generic Arithmetic Components, Low Power Design, Modulo 2^n–1 Addition, Residue Arithmetic.

1. Introduction

Abstract studies in computer arithmetic, often lead to generic frameworks for concrete implementations, where a design engineer may tailor the abstract design by replacing the generic parts with the best available, functionally equivalent and standard, components meeting his/her particular design goals (e.g., area, time, and power requirements). Computer arithmetic algorithms show variant performance potentials on alternative input and output encodings (e.g., redundant vs. nonredundant, and 2's complement vs. sign magnitude). Standard adder cells and components, as basic building blocks for implementing generic addition schemes, include full/half adder cells, carry-propagate adders (whether ripple-carry or carry accelerating adders), and carry-save adders (CSA) [1]. These standard components are periodically enhanced in performance through evolving new designs and implementations for basic cells such as full/half adders and carry-look-ahead blocks. Therefore, any arithmetic unit, generic in design, can be easily up-graded with new technological advances.

Modulo 2^n–1 addition [2, 3], solely or as the last stage of modular multiplication, independently or as a member of multi-channel arithmetic units based on residue number systems (RNS), is used in typical applications of modular arithmetic. Examples include RNS based digital signal processing [4, 5], FIR digital filters [6, 7, 8, 9], cryptography [10], fault tolerant computing [11] and error correction [12]. RNS arithmetic is popular, where most computations are done by multiplication, addition, and subtraction [13].

In this paper, we describe a new representation for modulo 2^n – 1 numbers leading to a generic modulo 2^n – 1 addition algorithm that can be implemented by an *n*-bit CSA and a generic *n*-bit binary adder. A background review on

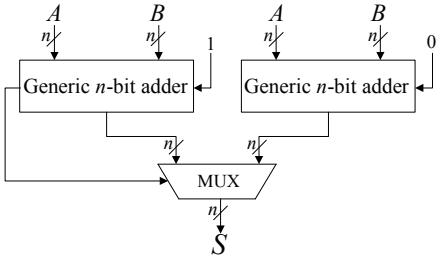
RNS and modulo 2^n±1 addition is given in Section 2, followed by a review of conventional representations for modulo 2^n–1 numbers and their addition algorithms in Section 3. Next in Section 4, we describe our novel stored negabit representation of modulo 2^n–1 numbers and the related generic modular adder design, where we show that a standard parallel prefix implementation of our algorithm leads to a latency of no more than that of the fastest adder for the conjugate modulo 2^n+1 [14]. Conversion from binary and the reverse conversion are discussed in Section 5. Finally, we draw our conclusions in Section 6.

2. Background

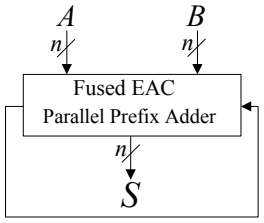
A residue number system Ψ is characterized by k integer moduli $m_{k-1}, \ldots m_1, m_0$. An integer value x ($\alpha \le x \le \beta$) is uniquely represented by $(x_{k-1}, \ldots x_1, x_0)$, where $x_i = x$ modulo m_i ($0 \le i \le k-1$), and $[\alpha, \beta]$ is the dynamic range of Ψ, whose cardinality $M_\Psi = \beta - \alpha + 1$. To maximize M_Ψ, the moduli are chosen to be pair-wise prime, such that $M_\Psi = m_{k-1} \times \ldots \times m_0$. The following equation shows a computation $x \bullet y$ as k parallel modular operations, where $\bullet$ is an arithmetic operator (e.g., + or ×), $y = (y_{k-1}, \ldots y_1, y_0)$, and subscript m_i ($0 \le i \le k-1$) stands for modulo m_i operation.

$$x \bullet y = ((x_{k-1} \bullet y_{k-1})_{m_{k-1}}, \ldots (x_1 \bullet y_1)_{m_1}, (x_0 \bullet y_0)_{m_0})$$

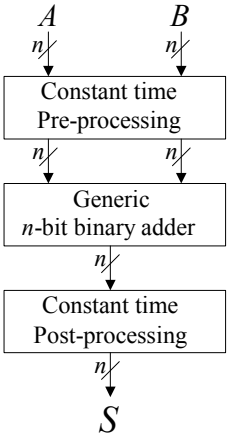
In RNS addition, for each operation channel associated to each moduli m_i ($0 \le i \le k-1$), the two residues are added modulo m_i. The latter, conceptually, consists of two operations: adding the residues, and the modulo operation on the sum. The moduli are chosen such that $\lceil \log m_i \rceil$ are almost the same for all i, leading to nearly equal addition speeds in all channels. Otherwise the overall addition latency would be, at best, in the order of $\lceil \log (\max (m_{k-1}, \ldots m_1, m_0)) \rceil$; not a constant-time operation in either case, but much faster than the addition of full-word binary representation of x and y. However, the latency of modulo operation may vary for different moduli, even though all their residues may be represented with the same number of bits. Therefore, one chooses the moduli such that the modulo operation is simplest. The first choice is a power of two (e.g., 2^n), leading to the fastest modulo operation, which is actually performed by ignoring the carry-out signal of the residue adder and keeping the sum. Next choices are primarily 2^n±1 [2, 15, 3, 16]. The three moduli have no common factor and the required dynamic range can be met by choosing the proper value for n. Unlike the 2^n case, the modulo operation for moduli 2^n±1 may involve substantive overhead in terms of hardware and latency, but yet much less than similar operation for other moduli with further distances from a power of two. Among the three popular moduli (i.e., 2^n and 2^n±1), modulo 2^n+1 residues are normally represented by n+1 bits, leading to slightly longer addition delay. Many researchers have offered variants of modulo 2^n±1 addition schemes (e.g., [17, 4, 2, 3, 16, and 18]), where the main focus is on delay optimized designs. The design with the least delay, for the usually slower modulo 2^n+1 adder, has been reported in [14], with a latency equal to that of $6 + 2\lceil \log n \rceil$ unit gates, using the delay estimation model in [19]. A similar work, for modulo $2^n - 1$ adder, adds two residues in $3 + 2\lceil \log n \rceil$ unit gate delay time [15].



a-Ccompound



b-Fused end-around carry



c- One step Generic

Figure1. Variants of modular adders

Zimmermann [2] offers a thorough analysis of different possible modulo 2^n±1 arithmetic implementations appeared in the literature. Modulo 2^n–1 addition, in particular, is either performed in two addition cycles, or with two parallel adders (Fig. 1a), sacrificing time and area, respectively. To lower the power dissipation, one of the two parallel adders of Fig. 1a, producing the unused result, may be switched off by fast carry-out prediction logic [20]. However a possible addition scheme, performing only one n-bit-long addition, may lead to power and area efficiency. For example, the modulo 2^n–1 adder of [15], with $3 + 2\lceil \log n \rceil$ delay, uses an overloaded parallel-prefix tree. The end-around carry increment cycle is fused in the tree, via 2^n crossing connections (See Fig. 1b, and its detailed version in Fig. 2, borrowed from [15]). Moreover, all the parallel prefix buffering nodes (see white circles in Fig. 6) are replaced by the more complex computing nodes (see black circles in Fig. 2). Nevertheless, the employed technique is beneficial only for parallel prefix adders and possibly other carry look-ahead architectures (i.e., it cannot be generalized for other addition schemes such that only one n-bit long addition takes place).

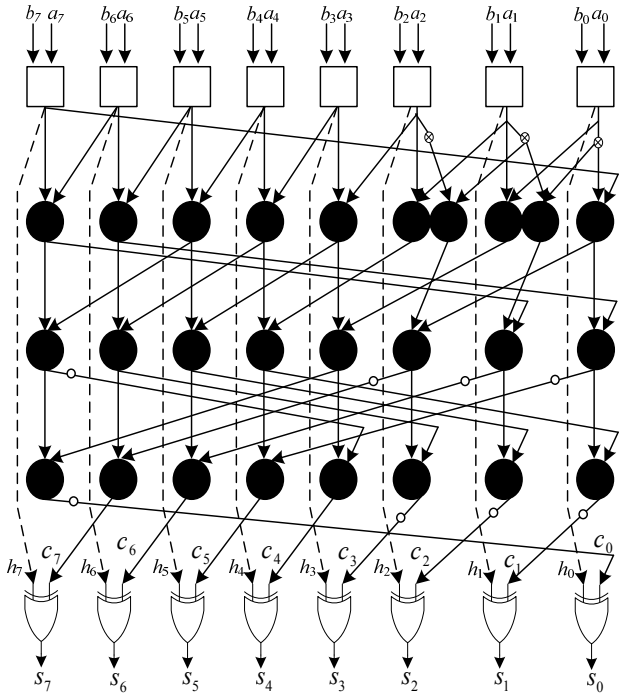


Figure 2. (adapted from [15]). A parallel prefix modulo 2^n–1 adder (n = 3), with 2^n crossing connections

The latter observation, and following our previous work on modulo $2^n + 1$ stored posibit adders [25], motivated us to develop a generic modulo 2^n–1 addition algorithm (see Section 4) that is independent of the embedded n-bit binary addition scheme. In this way, depending on the variant design goals (e.g., time, area, and power requirements), the designer may choose the best fitted addition technique (Figure 1c).
For a better comparison of different design efforts on modulo 2^n–1 addition, we define generic n-bit adders and one-step modular adders as follows:

Definition 1 (Generic *n*-bit adder): An n-bit nonredundant adder producing the sum with a latency δ which is not constant with respect to n, but at most linearly depends on n, is called a generic n-bit adder. ◄

There are different implementations for generic adders:

- **Sub-linear latency adders**: Carry look-ahead, parallel prefix, and conditional sum adders (O (log n) latency), carry-select or carry-skip adders (O ($\sqrt{n}$) latency) [1].
- **Linear latency adders**: Ripple-carry adders.

Definition 2 (One-step modular adder): A modular adder with only one generic n-bit adder in its critical delay path is called a one-step modular adder (e.g., all the ones Figure 1). ◄

3. Review of modulo 2^n – 1 addition

A straight forward algorithm for modulo 2^n – 1 addition may be described as follows:

Algorithm 1 (Pre-increment and post-decrement modulo 2^n − 1 addition):

Inputs: $A = a_{n-1}\ldots a_1\ a_0$, $B = b_{n-1}\ldots b_1\ b_0$ ($0 \le A, B < 2^n - 1$).
Output: $S = s_{n-1}..s_1 s_0 = (A+B)_{2^n-1}$ ($0 \le S < 2^n - 1$).

I. Compute $W = A + B + 1$ ($W = c_n\ w_{n-1}\ \ldots\ w_1\ w_0$).
II. If $c_n = 1$ then $S = w_{n-1}\ \ldots\ w_1\ w_0$ else $S = W - 1$. ◄

Step II is justified as:

- $c_n = 1$ (i.e., $A+B \ge 2^n - 1$):
$S = (A+B)_{2^n-1} = A + B - (2^n - 1) = W - 2^n = w_{n-1}\ldots w_1 w_0$
- $c_n = 0$ (i.e., $A+B < 2^n - 1$):
$S = (A+B)_{2^n-1} = A + B = W - 1$

Another algorithm, with end around carry addition, is:

Algorithm 2 (Modulo 2^n−1 end-around carry addition):

Inputs: $A = a_{n-1}\ldots a_1\ a_0$, $B = b_{n-1}\ldots b_1\ b_0$ ($0 \le A, B < 2^n - 1$), where 0 is represented by either n 0s, or n 1s.
output: $S = s_{n-1}..s_1 s_0 = (A+B)_{2^n-1}$ ($0 \le S < 2^n - 1$).

I. Compute $W = A + B = c_n\ w_{n-1}\ \ldots\ w_1\ w_0$.
II. $S = w_{n-1}\ \ldots\ w_1\ w_0 + c_n$ ◄

Step II is justified as:

- $c_n = 1$: $A + B \ge 2^n$
$\Rightarrow S = (A+B)_{2^n-1} = A + B - 2^n + 1 = (A + B + c_n)_{2^n}$
- $c_n = 0$:
  - $A + B < 2^n - 1$
  $\Rightarrow S = (A+B)_{2^n-1} = A + B = (A + B + c_n)_{2^n}$
  - $A + B = 2^n - 1$
  $\Rightarrow S = (A+B)_{2^n-1} = 0 = 11\ldots11 = (A + B + c_n)_{2^n}$

A commonly undesirable property of Algorithm 2 is the double representation for zero. In both algorithms, Step I involves an n-bit addition and Step II, in the worst case, goes through an n-bit increment/decrement operation. It is naturally desirable to find one-step algorithms that compute the required modulo addition by only one n-bit operation in the time span and unique representation of zero. Parallel computation of W and $W - 1$ ($W + 1$), is one common solution as depicted in Figure 1a. However, there is high penalty in area and power due to the extra active n-bit decrement (increment) logic.

Zimmermann [2] has enumerated four possible implementations for Algorithm 2. Three of the implementations require two cycles and the other one uses two n-bit adders in parallel. Efstathiou et. al. present a carry look-ahead implementation of Algorithm 2 [21] with only one n-bit addition. The trick, as shown below, is to postpone the computation involving c_{in} (i.e., end-around carry into the least significant position) to the last stage of positional carry derivations, thus avoiding a new carry look-ahead computation for enforcing the end-around carry.

$$c_i = G_{i-1} + P_{i-1}c_{in} = G_{i-1}, \quad c_n = G_{n-1},\ c_i^* = G_{i-1} + P_{i-1}\,\overline{G_{n-1}}$$

The carry-in signal c_i enters into position i ($c_0 = c_{in} = 0$), G

and P variables represent c_{in}-independent generate and propagate expressions (see equations below) based on positional generate and propagate signals (i.e., $g_i = a_i b_i$ $p_i = a_i + b_i$), and c_i^* is the final carry into position i used in $s_i = p_i \oplus c_i^*$.

$$G_i = g_{i-1} + p_{i-1} G_{i-2}, \qquad P_i = p_{i-1} P_{i-1}$$

Expressions for G_i and P_i (including G_{n-1}) depend on all a_j and b_j ($0 \le j < i$), and are computable in parallel by fundamental carry look-ahead operator cells or parallel prefix trees [22, 23] through the above recurrences.

Intermediate positional sum bits $w_i = p_i \oplus c_i^*$ need not actually be computed, as shown in direct derivation of s_i and c_i^*. Therefore the overall delay for sum bits s_i consists of the following partial delays, leading to $5 + 2\lceil \log n \rceil$ unit gate delays:

a. g_i and p_i computations: 1 unit gate
b. G and P derivations: 2 log n unit gates assuming 2-bit CLA blocks of a standard parallel prefix adder
c. c_i^* computation: 2 unit gates
d. The XOR for the final sum generation: 2 unit gates

A similar design appears in [15] that uses an overloaded parallel prefix adder, where the latency is reduced to $3 + 2\lceil \log n \rceil$ (i.e., the same as a standard parallel prefix adder with possible carry-in). The function of item c above is fused in all levels of the parallel prefix tree (Figure 2), which leads to replacement of all the simple buffering nodes with complex computing nodes.

This latency improvement is certainly valuable for independent single channel modulo $2^n - 1$ addition, or where there are multi channels with moduli of the form $2^{2^p - q} - 1$ ($n = 2^p - q$, and $0 \le q < 2^p - 1$). The latter constraint on q leads to p-level parallel prefix tree for all channels; hence equal latency channels.

However, the 2 unit gate latency saving, due to fusion of item c above, is achieved in price of 2^n crossing interconnections and overloading the parallel prefix tree (see Figure 2). This latency reduction is not always appreciated in multi-channel implementations with the conjugate moduli of the form $2^{2^p - q} + 1$. The reason is that the latency of the fastest of such adders is $6 + 2\lceil \log n \rceil = 2p + 6$ [14].

4. New one-step algorithm for modulo $2^n - 1$ addition

Recalling the decrement operation in Step II of algorithm 1, observe that Step II can be reformulated as:

$$S = w_{n-1} \ldots w_1 w_0 - \overline{c_n}$$

Instead of the n-bit decrement operation, we can store $\overline{c_n}$ as a negatively weighted bit (negabit for short) in position 0 of S. This would serve as a second *lsb* of the result, thus avoiding the decrement. Therefore we need to design modular addition scheme for stored negabit modulo $2^n - 1$ numbers. Algorithm 3 presents the details, where the stored negabits are distinguished by uppercase, primed and subscripted letters.

Algorithm 3 (Stored negabit modulo $2^n - 1$ addition):

Inputs: $A = a_{n-1} \ldots a_1 a_0 + A'_0$, $B = b_{n-1} \ldots b_1 b_0 + B'_0$ ($0 \le A, B < 2^n - 1$), where A'_0 and B'_0 are stored negabits, and $a_0 A'_0 = b_0 B'_0 = 0$ to avoid double zero representation.

output: $S = s_{n-1} \ldots s_0 + \acute{S}_0 = (A + B)_{2^n - 1}$, where $\acute{S}_0$ is a stored negabit ($s_0 S'_0 = 0$, and $0 \le S < 2^n - 1$).

I. Compute $W = A + B + 1 = w_n w_{n-1} \ldots w_1 w_0$.
II. $S = w_{n-1} \ldots w_1 w_0 - \overline{w_n}$. ◀

Step I is composed of the following sub-steps:

a. Derive $A'_0 + B'_0 + 1$, and sign-extend the sum to derive $T = T_{n+1} t_n \ldots t_1 t_0$, where $T_{n+1} = t_n = \ldots = t_1 = A'_0 B'_0$, and $t_0 = \overline{A'_0 \oplus B'_0}$.

b. Carry-save add $a_{n-1} \ldots a_1 a_0$, $b_{n-1} \ldots b_1 b_0$, and $t_{n-1} \ldots t_1 t_0$.

c. Derive W through an n-bit ripple-carry or carry-accelerate addition. Note that, given $0 \le W \le 2^{n+1} - 1$, $w_{n+1} = 0$.

Now, Step II may be justified as:

- $w_n = 1$: $A + B + 1 \ge 2^n$, $S = A + B - 2^n + 1$, $S = w_{n-1} \ldots w_1 w_0 - 0$.
- $w_n = 0$: $A + B + 1 < 2^n$, $S = A + B = w_{n-1} \ldots w_1 w_0 - 1$.

The latter decrement does not actually take place, instead $S'_0 = \overline{w_n}$ is stored as a negabit. Figure 3 depicts a symbolic illustration of Algorithm 3 and Figure 4 shows a block diagram of the adder.

		a_{n-1}	...	a_1	a_0
					A'_0
		b_{n-1}	...	b_1	b_0
					B'_0
					1
0	0	a_{n-1}	...	a_1	a_0
0	0	b_{n-1}	...	b_1	b_0
$-T_{n+1}$	t_n	t_{n-1}		t_1	t_0
$-T_{n+1}$	t_n	u_{n-1}	…	u_1	u_0
0	c_n	c_{n-1}	…	c_1	
w_{n+1}	w_n	w_{n-1}	…	w_1	w_0
		s_{n-1}	…	s_1	s_0
					S'_0

Figure 3. Stored negabit modulo $2^n - 1$ addition

The n-bit binary adder of Figure 4 is a generic one (see Def. 1), which may be replaced by any binary adder desired by the design engineer to meet the variant area, power, and latency requirement. For example a delay optimized design based on parallel prefix adder is depicted in Figure 6. Among all the c signals, only the carry signal of the dashed box, delivered after three unit gates, is on the critical delay path. Therefore, assuming the latency of a full adder to be equal to that of two unit gates, the preprocessing (i.e., derivation of t signals and carry-save addition) imposes a delay of three unit gates. The rest of the critical path is exactly the same as in Figure 2 with a delay of $2\lceil \log n \rceil + 3$, totaling the addition delay of Figure 6 to $2\lceil \log n \rceil + 6$. The black circles exactly represent the same logic as the ones in Figure 2. However the

white circles are merely simple buffering units to prevent high fan-out. Figure 5 depicts the inside of these circles.

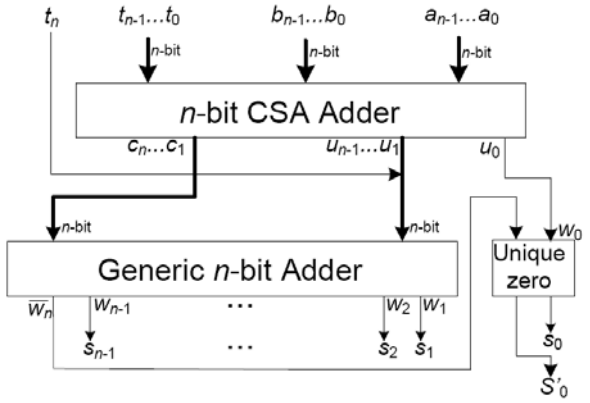


Figure 4. Stored negabit modulo $2^n - 1$ Adder

To prevent double zero representation of results, due to possibility of $s_0 = S'_0 = 1$, we have replaced the XOR gate in the most significant position with a new box (see the left-bottom box in Figure 6) implementing the following equations, where $h = t_n \oplus c_n$, and c is the carry into position n:

$$s_0 = u_0\, h\, \bar{c} + u_0\, \bar{h}\, c, \qquad S'_0 = \overline{u_0 + h}\, \bar{c} + \overline{u_0}\, h\, c$$

The combination $s_0 = S'_0 = 1$, would never occur through the latter equations; hence unique zero representation is achieved only in price of few more gates and without any extra delay. The reason is that the least significant bits s_0 and S'_0 are available right after a multiplexer controlled by c. This is the same time as those of other sum bits (i.e., just after an XOR delay with respect to carry signals).

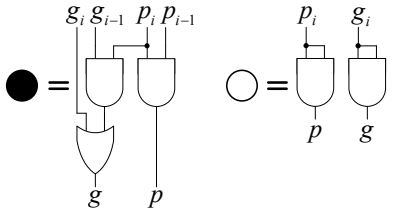


Figure 5. The logic inside of the circles of Figure 6

The desirable properties of the adder of Figure 6 are:

- Equal latency with the best conjugate modulo $2^n + 1$ adder of [14]
- Unique zero representation
- Lack of 2^n extra crossing interconnections compared to Figure 2
- Standard parallel prefix tree with several simple buffering nodes instead of complex computing nodes
- Lower power dissipation than that of compound adder of Figure 1a

The latter advantage is due to less switching activity of the carry-save adder of Figure 3 than that of any of the two n-bit adders of Figure 1a. In [15], for prohibition of double zero representation, a logic is proposed that is scattered within all data paths, while ours uses only few gates in the most significant position. The advantage of lack of 2^n crossing interconnections is considerable. However, unlike the advantage of unique zero representation, the latter property is not an absolute advantage, for in the design of Figure 2, the crossings could also be avoided by introducing direct end-around carry fusion logic with 2 unit-gates extra latency.

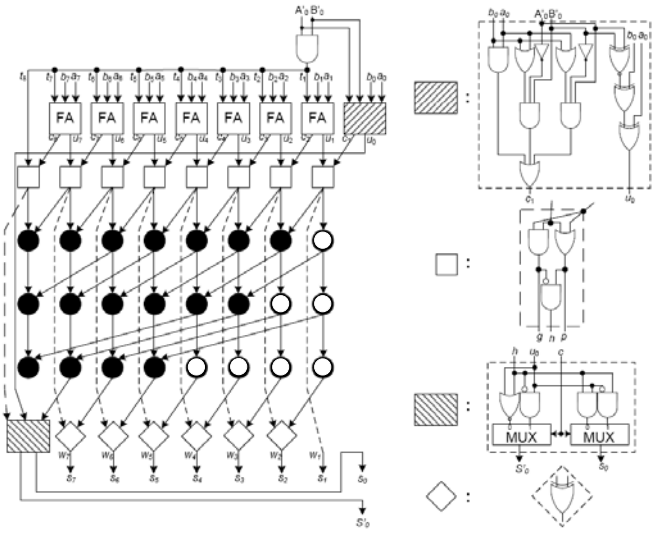


Figure 6. Stored negabit parallel prefix modulo 2^n–1 adder

5. Conversion from/to binary

Conversion of a binary number I to stored negabit representation of $(I)_{2^n-1}$ follows the conventional binary to modulo 2^n–1 conversion, and insertion of a 0-valued negabit as the second *lsb* of the residue.

For the reverse conversion, the second *lsb* does not introduce considerable inefficiency. Residue to binary converters, normally implement the Chinese remainder theorem [13] using carry save adders (CSA) for the required multi-operand binary addition. With a careful design of the CSA tree, contribution of the second *lsb* to the overall conversion delay may only add one level of CSA. For example, for the popular moduli set $\{2^n \pm 1, 2^n\}$, the following Equation-set adapted from [24] converts (X, Y, Z) to I, where $X = (I)_{2^n} = x_{n-1}...x_0$, $Y = (I)_{2^n-1} = y_{n-1}...y_0 + Y'_0$, and $Z = (I)_{2^n+1} = z_n\, z_{n-1}...z_0$:

$$I = 2^n I' + X,$$
$$I' = (-(2^n X + Z) + 2^{n-1}(2^n + 1)(Y + Z))_{2^{2n}-1}$$

To handle the negative term $-(2^n X + Z)$ and the negabit Y'_0, we do the following derivations modulo 2^{2n}–1, where $\overline{A}$ stands for one's complements of A, and $\hat{Y} = Y - Y'_0 = y_{n-1}...y_0$.

$$-(2^n X + Z) - 2^{n-1}(2^n + 1)Y'_0 = 2^{2n} - 1 - 2^n X - Z - 2^{n-1}(2^n + 1)(1 - \overline{Y'_0}) = 2^n(2^n - 1 - X + 1) - 2^{n+1} + 2^{n+1} - 1 - Z + 2^{n-1}(2^n + 1)\overline{Y'_0} - 2^{n-1}(2^n + 1) = 2^n\overline{X} + \overline{Z} + 2^{2n-1}\overline{Y'_0} + 2^{n-1}\overline{Y'_0} + 2^{2n-1} - 2^n - 2^{n-1} - 1$$

Replacing the latter in the equation for I' leads to:

$$I' = \left(2^n\overline{X} + \overline{Z} + 2^{2n-1}\left(\hat{Y} + \overline{Y'_0} + Z\right) + 2^{n-1}\left(\hat{Y} + \overline{Y'_0} + Z\right) + 2^{2n-1} - 2^n - 2^{n-1} - 1\right)_{2^{2n}-1}$$

Figure 7 shows how the latter can be computed as a 2*n*-bit six-operand addition, where the gray parts correspond to the term $2^{2n-1}(Y+Z)$. The number of operands for standard case (i.e., without stored negabit) would be five. Both cases require three levels of carry-save adders and a final carry-propagate adder to evaluate the sum.

2^{2n-1}	2^{2n-2}	…	2^{n+1}	2^n	2^{n-1}	2^{n-2}	…	2^0
$\overline{x_{n-1}}$	$\overline{x_{n-2}}$	…	$\overline{x_1}$	$\overline{x_0}$	$\overline{z_{n-1}}$	$\overline{z_{n-2}}$	…	$\overline{z_0}$
z_n	y_{n-1}	…	y_2	y_1	y_0			
y_0				$\overline{z_n}$	z_0	y_{n-1}	…	y_1
z_0	z_{n-1}	…	z_2	z_1	z_n	z_{n-1}	…	z_1
$\overline{Y_0'}$					$\overline{Y_0'}$			
1	1	…	1	0	0	1	…	1

Figure 7. Multi-operand addition to compute I'

6. Conclusion

We introduce the stored negabit representation of modulo $2^n - 1$ residues, with unique zero representation property, leading to a generic modulo $2^n - 1$ addition algorithm that is implemented by:

I. A constant time preprocessor composed of:
   - A sign-extended negabit half-adder, and
   - An *n*-bit carry-save adder

II. A generic *n*-bit adder

III. A postprocessor to restore unique zero representation

The generic design provides the design engineer with the opportunity of replacing each generic part (e.g., the CSA in Step I, or the adder of Step II above) by any available, standard and functionally equivalent, component best meeting the design goals (e.g., time, area, or power requirements). Possible upgrades due to future technological advances, in standard computer arithmetic components, are easy to achieve. For example we replace the generic *n*-bit adder by a standard parallel prefix adder for a delay optimized design, leading to $6 + 2\lceil \log n \rceil$ unit-gate latency, equal to the best result for the conjugate modulo $2^n + 1$ adders [14]. We show advantages of the latter to a previously reported [15] specific parallel prefix modulo $2^n - 1$ adder with $3 + 2\lceil \log n \rceil$ unit-gate latency, but with the VLSI-unfriendly 2^n backward interconnections, and provisions for unique zero representation in all data paths.

Our generic design, in comparison with the conventional generic design with two parallel *n*-bit adders, has the advantage of use of a low-power CSA instead of one of the *n*-bit adders. Furthermore, it was shown that the new representation does not increase the complexity of binary to residue and the reverse conversions. Research is ongoing for designing of modulo 2^n–1 multipliers based on stored negabit representation of residues.

Acknowledgment

This research was funded by Shahid Behesti University. The Author wishes to thank Amir Kaivani and Saied Gorgin for their comments and help during the preparation of this manuscript.

References

[1] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press, 2010.

[2] R. Zimmerman, "Efficient VLSI Implementation of Modulo 2^n–1 Addition and Multiplication," *Proc. of the IEEE Symposium on Computer Arithmetic*, pp. 158-167, 1999.

[3] C. Efstathiou, H. T. Vergos, and D. Nikolos, "On the Design of Modulo 2^n – 1 Adders," *Proc. of the International Conference on Electronics, Circuits and Systems*, pp. 517-520, 2001.

[4] M. Bhardwaj, A. B. Premkumar, and T. Srikanthan, "Breaking the 2*n*-bit Carry Propagation Barrier in Residue to Binary Conversion for the {2^n–1, 2^n, 2^n+1} Module Set," *IEEE Transactions on Circuits and Systems*–II, vol. 45, no. 9, pp. 998-1002, 1998.

[5] P. G. Fernandez, A. Garcia, J. Ramirez, L. Parrilla, and A. Lloris, "A RNS-Based Matrix-Vector-Multiply FCT Architecture for DCT Computation," *Proc. of the IEEE Midwest Symposium on Circuits and Systems*, pp. 350-353, 2000.

[6] R. Conway and J. Nelson, "Improved RNS FIR Filter Architectures," *IEEE Transactions on Circuits and Systems-II: Express Briefs*, vol. 51, no. 1, pp. 26-28, 2004.

[7] W. L. Freking and K. K. Parhi "Low-Power FIR Digital Filters using Residue Arithmetic," *Proc. of the Asilomar Conference on Signals, Systems and Computers*, pp. 739–43, 1997.

[8] M. K. Ibrahim, "Novel Digital Filter Implementations Using Hybrid RNS-Binary Arithmetic," *Signal Processing*, vol. 40, no. 2-3, pp. 287-294, 1994.

[9] B. Parhami, "Novel Digital Filter Implementations Using Hybrid RNS-Binary Arithmetic," *Signal Processing*, Vol. 51, no. 2-3, pp. 65-67, 1996.

[10] S. Yen, S. Kim, S. Lim, and S. Moon, "RSA Speedup with Chinese Remainder Theorem Immune against Hardware Fault Cryptanalysis," *IEEE Transactions on Computers*, vol. 52, no. 2, pp. 461-472, 2003.

[11] E. Kinoshita and K. Lee, "A Residue Arithmetic Extension for Reliable Scientific Computation," *IEEE Transactions on Computers*, vol. 46, no. 2, pp. 129-138, 1997.

[12] L. Yang and L. Hanzo, "Redundant Residue Number System Based Error Correction Codes," *Proc. of the IEEE Vehicular Technology* Conference, pp. 1472-1476, 2001.

[13] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Taylor, *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*, IEEE Press, 1986.

[14] C. Efstathiou, H. T. Vergos, and D. Nikolos, "Fast Parallel-Prefix 2^n+1 Adder," *IEEE Transactions on Computers*, vol. 53, no. 9, pp. 1211-1216, 2004.

[15] L. Kalamboukas, D. Nikolos, C. Efstathiou, H. T. Vergos, and J. Kalamatianos, "High-Speed Parallel-Prefix Modulo 2^n–1 Adders," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 673-680, 2000.

[16] H. T. Vergos, C. Efstathiou, D. Nikolos, "Diminished-One Modulo 2^n + 1 Adder Design," *IEEE Transactions on Computers*, vol. 51, no. 12, pp. 1389-1399, 2002.

[17] A. Ashur, M. K. Ibrahim, and A. Aggoun, "Novel RNS Structures for the Moduli Set {2^n–1, 2^k, 2^k+1} and their Application to Digital Filter Implementation," *Signal processing*, vol. 46, no. 3, pp. 331-343, 1995.

[18] C. Efstathiou, H. T. Vergos, and D. Nikolos, "Modulo 2n 1 Adder Design using Select-Prefix Blocks," *IEEE Transactions on Computers*, vol. 52, no. 11, pp. 1399–1406, 2003.

[19] A. Tyagi, "A Reduced-Area Scheme for Carry-Select Adders," *IEEE Transactions on Computers*, vol. 42, no. 10, pp. 1163-1170, 1993.

[20] A. Nannarelli, G. C. Cardallili, and M. Re, "Power-Delay Tradeoffs in Residue Number System," *Proc. of the IEEE International Symposium on Circuits and Systems*, pp. 413-416, 2003.

[21] C. Efstathiou, D. Nikolos, and J. Kalamatianos, "Area-Time Efficient Modulo 2^n – 1 Adder Design," *IEEE Transactions on Circuits and Syst*ems, vol. 41, no. 7, pp 463-467, 1994.

[22] P. M. Kogge and H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Transactions on Computers*, vol. 22, no. 8, pp. 786-792, 1973.

[23] R. E. Ladner, and M. J. Fischer, "Parallel Prefix Computation," *Journal of.ACM*, vol. 27, no. 4, pp. 831–838, 1980.

[24] Z. Wang, G. A. Jullien, and W. C. Miller, "An Improved Residue-to-Binary Converter," *IEEE Transactions on Circuits and Systems I: Fundamental* Theory *and Applications*, vol. 47, no. 9, pp. 1437-1440, 2000.

[25] G. Jaberipur, "A One-Step Modulo 2^n+1 Adder Based on Double-lsb Representation of Residues," *The CSI Journal on Computer Science and Engineering*, vol. 4, no. 2-4, pp. 10-16, 2006.

Ghassem Jaberipur received his B.S in Electrical Engineering and PhD in Computer Engineering from Sharif University of Technology in 1974 and 2004, respectively, his M.S in Engineering (majoring in computer hardware) from UCLA in 1976, and his M.S in Computer Science from University of Wisconsin in Madison in 1979. He is currently an associate professor of computer engineering in the department of Electrical and Computer Engineering in Shahid Beheshti University (Tehran, Iran). His main activities include teaching undergraduate courses in theory and implementation of programming languages, and graduate courses in compiler construction and computer arithmetic. His main research interest is in computer arithmetic.

E-mail: Jaberipur@sbu.ic.ir

Paper Handling Data:

Submitted: 08.13.2008
Received in revised form: 09.18.2010
Accepted: 12.13.2010
Corresponding author: Dr. Ghassem Jaberipur,
Department of Electrical and Computer Engineering, Shahid Beheshti University, Tehran, Iran.