Adder Based Residue to Binary Number Converters for $(2^n - 1, 2^n, 2^n + 1)$

Yuke Wang, Xiaoyu Song, Mostapha Aboulhamid, *Member, IEEE*, and Hong Shen

***Abstract*—Based on an algorithm derived from the New Chinese Remainder Theorem I, we present three new residue-to-binary converters for the residue number system $(2^n - 1, 2^n, 2^n + 1)$ designed using $2n$-bit or n-bit adders with improvements on speed, area, or dynamic range compared with various previous converters. The $2n$-bit adder based converter is faster and requires about half the hardware required by previous methods. For n-bit adder-based implementations, one new converter is twice as fast as the previous method using a similar amount of hardware, whereas another new converter achieves improvement in either speed, area, or dynamic range compared with previous converters.**

***Index Terms*—Adders, algorithm, arithmetic, circuit, residue number system.**

I. Introduction

THERE has been interest in residue number system (RNS) arithmetic as a basis for computational hardware since the 1950s [1], [2]. During the past decade, the RNS has received considerable attention in arithmetic computation and signal processing applications, such as fast Fourier transforms, digital filtering, and image processing [2], [3]. The main reasons for the interests are the inherent properties of RNS such as parallelism, modularity, fault tolerance, and carry-free operations [3]. The technology advantages offered by VLSI have added a new dimension to the implementation of RNS-based architectures. Several high-speed VLSI special-purpose digital signal processors have been successfully implemented.

The two most important issues for the residue arithmetic are the choice of moduli sets and the conversion of the residue to binary numbers. The residue number system based on the set of moduli $(2^n-1, 2^n, 2^n+1)$ has gained popularity and is expected to play an increasing role in RNS digital signal processing [5]. For general moduli sets, the residue to binary conversions are traditionally based on the Chinese Remainder Theorem (CRT) or mixed-radix conversion. Some new general conversion algorithms called New Chinese Remainder Theorems have been recently proposed with smaller size modulo operations [13], [14].

Manuscript received August 5, 1999; revised March 19, 2002. The associate editor coordinating the review of this paper and approving it for publication was Prof. Scott C. Douglas.

Y. Wang is with the Department of Computer Science, Erik Jonsson School of Engineering and Computer Science, University of Texas at Dallas, Richardson, TX 75083-0688 USA (e-mail: Yuke@utdallas.edu).

X. Song is with the Department of Electrical and Computer Engineering, Portland State University, Portland, OR 97207-0751 USA.

M. Aboulhamid is with the Departement d'informatique et de Recherche Operationnelle, Universite de Montreal, Montreal, QC H3C 3J7 Canada.

H. Shen is with the School of Information Science, Japan Advanced Institute of Science and Technology, Tatsunokuchi, Ishikawa, Japan.

Publisher Item Identifier S 1053-587X(02)05635-0.

Several conversion methods for $(2^n - 1, 2^n, 2^n + 1)$ have been reported [6]–[11], [15]–[18]. Early converters [17] for such moduli sets use ROM, which can be limited by the size n. In recent years, converters using $2n$-bit or n-bit adders have been proposed. These converters are designed using special formulas rather than the general CRT algorithm, and improvement in terms of hardware complexity has been reported. Detailed comparisons of all those converters are presented in Tables I and II.

In this paper, for the moduli set $(2^n - 1, 2^n, 2^n + 1)$, we present new and uniform algorithms designed using the New Chinese Remainder Theorems for the RNS to binary conversion. Three different converters using either $2n$-bit or n-bit adders are proposed. The $2n$-bit adder-based converter is faster and requires about half the hardware required by the previous methods [7]–[9]. For n-bit adder-based implementations, one new converter is twice as fast as the previous method [6] using a similar amount of hardware, whereas another new converter achieves improvement in both speed and area. The amount of hardware for the new converters is similar for the n-bit adder-based converter compared with the one in [9]. However, in [9], not the entire dynamic range of numbers is used.

In the following, we first introduce background material and derive the formulas; then, we show an example and propose three different hardware implementations.

II. Background

For any two numbers X and P_i, $x_i = X \bmod P_i$ is defined as $X = x_i + bP_i$ for some integer b such that $0 \le x_i < P_i$. $X \bmod P_i$ can be written as X_{P_i} or $|X|_{P_i}$.

An RNS is defined in terms of a set of relatively prime moduli $(P_1, P_2, \ldots, P_k)$, where $GCD(P_i, P_j) = 1$ for $i \neq j$. A binary number X can be represented as $X = (x_1, x_2, \ldots, x_k)$, where $x_i = X \bmod P_i$, $0 \le x_i < P_i$. Such a representation is unique for any integer $X \in [0, M)$, $M = \prod_{1 \le i \le k} P_i$.

For the RNS defined on the moduli set $(2^n-1, 2^n, 2^n+1)$, a binary number $X \in [0, 2^n(2^n-1)(2^n+1)) = [0, 2^{3n}-2^n)$ can be represented as a tuple (x_1, x_2, x_3), where x_1 and x_2 are two n-bit binary numbers; x_3 is a $n+1$-bit binary number, which is denoted as

$$\begin{aligned} X_{2^n-1} &= X \bmod (2^n - 1) = x_1 \\ &= x_{1(n-1)} x_{1(n-2)} \cdots x_{11} x_{10} \qquad (1) \\ X_{2^n} &= X \bmod 2^n = x_2 \\ &= x_{2(n-1)} x_{2(n-2)} \cdots x_{21} x_{20} \qquad (2) \\ X_{2^n+1} &= X \bmod (2^n + 1) = x_3 \\ &= x_{3n} x_{3(n-1)} x_{3(n-2)} \cdots x_{31} x_{30}. \qquad (3) \end{aligned}$$

To convert a residue number $(x_1, x_2, \ldots, x_n)$ into its binary number X, the CRT and mixed-radix conversion method are

TABLE I
PERFORMANCE COMPARISON OF $2n$-BIT ADDER-BASED CONVERTERS

Converter	FAs	AND /OR	XOR/ XNOR	CLAs -2n	other	Delay
[8][11]	6n	-	n+1	2	-	$2t_{CPA(n)} + 2t_{CPA(2n)} + 2t_{XOR}$
[9]	6n	4n-2	2n	1	-	$3t_{CPA(2n)} + t_{XOR} + \lceil \log(2n) \rceil t_{AND}$
[18]	6n	n+3	n+1	1		$2t_{FA} + t_{inv} + t_{CPA(2n)} + t_{MUX}$
[7]-CE	4n	2n-1	2n	1	2n+1 inverter	$2t_{FA} + t_{inv} + 2t_{CPA(2n)} + 3t_{MUX}$
ConverterI	2n	-	1	1	1HA, 2MUX 2n+1 inverter	$t_{inv} + t_{MUX} + t_{FA} + 2t_{CPA(2n)}$

TABLE II
PERFORMANCE COMPARISON OF n-BIT ADDER-BASED CONVERTERS

	FA	MUX	XOR	AND /OR	INV	HA	Mem	CLA	Delay
CII	2n	2n+2	1	2	2n+5	1	0	4	$t_{inv} + t_{FA} + t_{MUX} + t_{CLA(n)}$
CIII	2n	2n+2	2n-1	2+2n	2n+7	1	0	2	$t_{inv} + t_{FA} + t_{XOR} + t_{AND} + t_{MUX} + t_{CLA(n)}$
[6]	n	0	2	10	4n+8	0	8n	4	$3t_{inv} + t_{FA} + t_{XOR} + t_{AND} + 2t_{CLA(n)}$
[18]	6n	2n	n+1	n+3	0	0	0	4	$2t_{FA} + 2t_{NAND} + t_{CPA(n)} + t_{MUX}$
[15]	2n	2n	0	0	2n	0	0	4	$t_{OR} + t_{ANDOR} + t_{inv} + 2t_{MUX} + t_{FA} + t_{CPA(n)}$

traditionally used. We define $\left|P_i^{-1}\right|_{P_j}$ to be the multiplicative inverse of $P_i \bmod P_j$, i.e., $\left|P_i^{-1}\right|_{P_j} {}^* P_i = 1 \bmod P_j$.

Chinese Remainder Theorem: The binary number X is computed by $X = \left|\sum_{i=1}^{n} N_i \left|N_i^{-1}\right|_{P_i} x_i\right|_M$, where $N_i = M/P_i$, and $\left|N_i^{-1}\right|_{P_i}$ is the multiplicative inverse of $N_i \bmod P_i$.

The CRT requires a modulo M (large-valued) operation, which is not very efficient. Therefore, the converters proposed in [6]–[11], [15], [16], and [18] use specially designed algorithms to remove the modulo M operation or to reduce the size of the modulo operation. For example, the converters in [6] and [14] are based on the formula $X = D_2{}^*2^{2n} + D_1{}^*2^n + x_2$, and methods are required to compute the coefficients D_1 and D_2. In [7], [9], and [15], the converters are based on the formula $X = Y_{(2^{2n}-1)} + x_2$, and methods for computing Y are needed in each paper. In [7], the number Y is calculated as $|A + B + C - Z|_{2^{2n}-1}$, where A, B, C, and Z are $2n$-bit numbers obtained from (x_1, x_2, x_3). On the other hand, the third formula in [15] reduces the size of the modulo operation from M to N_i at the expense that some part of the dynamic range $X \in [0, M)$ will not be useable.

Recently, some alternative general conversion algorithms [the New Chinese Remainder Theorems (New CRT-I, II, and III) [13], [14]] have been proposed, which reduce the size of the modulo operation required by the CRT.

New Chinese Remainder Theorem I (New CRT-I): Given the residue number $(x_1, x_2, \ldots, x_n)$, the binary number X can be computed by (4), shown at the bottom of the page, which can be easily simplified as (5), shown at the bottom of the page, where $k_1 P_1 = 1 \bmod P_2 \cdots P_n$, $k_2 P_1 P_2 = 1 \bmod P_3 \cdots P_n, \ldots, k_{n-1} P_1 \cdots P_{n-1} = 1 \bmod P_n$.

Based on the New CRT-I, we have the following theorem for $n = 3$.

Theorem 1: For a three moduli set (P_1, P_2, P_3), the binary number $X = (x_1, x_2, x_3)$ can be calculated as

$$X = x_1 + |k_1(x_2 - x_1) + k_2 P_2 (x_3 - x_2)|_{P_2 P_3} P_1 \quad (6)$$

where $k_1 P_1 = 1 \bmod P_2 P_3$, and $k_2 P_1 P_2 = 1 \bmod P_3$.

In Section III, we apply (6) to the moduli set $(2^n - 1, 2^n, 2^n + 1)$ to design the residue to binary converters.

III. BASIC FORMULAS

The following Theorem 2 is a direct application of Theorem 1.

Theorem 2: For the moduli set $(2^n - 1, 2^n, 2^n + 1)$, the number X can be computed from (x_1, x_2, x_3) by the formula

$$X = x_2 + 2^n {}^*|(x_2 - x_3) + (x_1 - 2x_2 + x_3) 2^{n-1} (2^n + 1)|_{(2^{2n}-1)}. \quad (7)$$

$$X = |x_1 + k_1 P_1 (x_2 - x_1) + k_2 P_1 P_2 (x_3 - x_2) + \cdots + k_{(n-1)} P_1 P_2 \cdots P_{n-1} (x_n - x_{n-1})|_{P_1 P_2 \cdots P_{n-1} P_n} \quad (4)$$

$$X = x_1 + P_1 |k_1(x_2 - x_1) + k_2 P_2 (x_3 - x_2) + \cdots + k_{n-1} P_2 \cdots P_{n-1} (x_n - x_{n-1})|_{P_2 \cdots P_{n-1} P_n} \quad (5)$$

Proof: Using Theorem 1 and assuming that $P_1 = 2^n$, $P_2 = 2^n + 1$, and $P_3 = 2^n - 1$, we have $k_1 = 2^n$ and $k_2 = 2^{n-1}$ such that $k_1 2^n = 1 \bmod (2^{2n} - 1)$ and $k_2 2^n (2^n + 1) = 1 \bmod (2^n - 1)$. Therefore, we have the first equation at the bottom of the page.

Proposition 1: For any integers a and b, we have $\lfloor a/2 \rfloor + b = \lfloor (a + 2b)/2 \rfloor$.

Proof: Letting $a_0 = a \bmod 2$, we have that

$$a = 2^* \left\lfloor \frac{a}{2} \right\rfloor + a_0,$$

$$a + 2b = 2^* \left\lfloor \frac{a}{2} \right\rfloor + a_0 + 2b = 2^* \left(\left\lfloor \frac{a}{2} \right\rfloor + b \right) + a_0$$

and therefore, we have the following proposition.

Proposition 2: X can be computed by (8.1)–(8.4), shown at the bottom of the page, where x_{10} and x_{30} are the least significant bits of x_1 and x_3, respectively, and $x_{10} \oplus x_{30}$ denotes the XOR operation, i.e., $x_{10} \oplus x_{30} = x_{10}$ XOR x_{30}.

Proof: Let $Y = |(x_2 - x_3) + (x_1 - 2x_2 + x_3)2^{n-1}(2^n + 1)|_{(2^{2n}-1)}$ in (7); then, we have $X = x_2 + 2^{n*}Y$, and $Y = |(x_2 - x_3) + (x_1 - 2x_2 + x_3)2^{n-1} + (x_1 - 2x_2 + x_3)2^{2n-1}|_{(2^{2n}-1)}$.

Since $(x_1 - 2x2 + x_3) = 2^* \lfloor (x_1 - 2x2 + x_3)/2 \rfloor + (x_1 - 2_{x_2} + x_3)_2$, we denote $z_0 = (x_1 - 2x_2 + x_3)_2 = (x_{10} \oplus x_{30})$, $z = \lceil (x_1 - 2x_2 + x_3)/2 \rceil$, and therefore, we have the last equation at the bottom of the page, i.e., we have $Y = \{A + 2^{n*}B\}_{(2^{2n}-1)}$. **Q.E.D.**

Next, we present an example using the above formulas.

Example: Consider the example shown in [6]. Let $(2^n - 1, 2^n, 2^n + 1) = (7, 8, 9)$ and a number 407, which can be represented as (1, 7, 2) in the moduli set (7, 8, 9).

Now, given (1, 7, 2) = (001, 111, 0010), we have the equation at the bottom of the next page. Compared with the long calculation on [6, p. 56], the above process is much simpler.

$$\begin{aligned} X &= x_2 + 2^n \left| 2^n(x_3 - x_2) + 2^{n-1}(2^n + 1)(x_1 - x_3) \right|_{2^{2n}-1} \\ &= x_2 + 2^n \left| 2^n(x_3 - x_2) + (2^{2n} - 1)(x_3 - x_2) + 2^{n-1}(2^n + 1)(x_1 - x_3) \right|_{2^{2n}-1} \\ &= x_2 + 2^n \left| (x_2 - x_3) + (2^{2n} + 2^n)(x_3 - x_2) + 2^{n-1}(2^n + 1)(x_1 - x_3) \right|_{2^{2n}-1} \\ &= x_2 + 2^n \left| (x_2 - x_3) + 2^{n-1}(2^n + 1)(x_1 - 2x_2 + x_3) \right|_{2^{2n}-1}. \end{aligned}$$

$$X = x_2 + 2^{n*}Y \tag{8.1}$$

$$\text{where } Y = |A + 2^{n*}B|_{(2^{2n}-1)} \tag{8.2}$$

$$A = \left\lfloor \frac{(x_1 + (x_{10} \oplus x_{30})^* 2^n) + (2^n - 1 - x_3) + (2^n - 1)}{2} \right\rfloor \tag{8.3}$$

$$B = \left\lfloor \frac{(x_1 + (x_{10} \oplus x_{30})^* 2^n) + x_3 + 2(2^n - 1 - x_2)}{2} \right\rfloor \tag{8.4}$$

$$\begin{aligned} Y &= \left| \left[(x_2 - x_3) + z^* 2^n + (x_{10} \oplus x_{30})^* 2^{n-1} + z^* 2^{2n} + (x_{10} \oplus x_{30})^* 2^{2n-1} \right] \right|_{(2^{2n}-1)} \\ &= \left| \left[(x_2 - x_3 + z) + (x_{10} \oplus x_{30})^* 2^{n-1} + z^* 2^n + (x_{10} \oplus x_{30})^* 2^{2n-1} \right] \right|_{(2^{2n}-1)} \\ &= \left| \left\{ \left\lfloor \frac{(x_1 - x_3)}{2} \right\rfloor + (x_{10} \oplus x_{30})^* 2^{n-1} + 2^{n*} \left\lfloor \frac{(x_1 - 2x_2 + x_3)}{2} \right\rfloor + (x_{10} \oplus x_{30})^* 2^{2n-1} \right\} \right|_{(2^{2n}-1)} \\ &= \left| \left\{ \left[\left\lfloor \frac{(x_1 - x_3)}{2} \right\rfloor + z_0{}^* 2^{n-1} \right] + 2^n \left[\left\lfloor \frac{(x_1 - 2x_2 + x_3)}{2} \right\rfloor + z_0{}^* 2^{n-1} \right] \right\} \right|_{(2^{2n}-1)} \\ &= \left| \left\{ \left(\left\lfloor \frac{(x_1 - x_3) + z_0{}^* 2^n}{2} \right\rfloor \right) + 2^n \left(\left\lfloor \frac{(x_1 - 2x_2 + x_3) + z_0{}^* 2^n}{2} \right\rfloor \right) \right\} \right|_{(2^{2n}-1)} \\ &= \left| \left\{ \left(\left\lfloor \frac{(x_1 - x_3) + z_0{}^* 2^n + 2^{n+1}}{2} \right\rfloor - 2^n \right) + 2^n \left(\left\lfloor \frac{(x_1 - 2x_2 + x_3 + 2^{n+1}) + z_0{}^* 2^n}{2} \right\rfloor - 2^n \right) \right\} \right|_{(2^{2n}-1)} \\ &= \left| \left\{ \left(\left\lfloor \frac{(x_1 - x_3) + z_0{}^* 2^n + 2^{n+1}}{2} \right\rfloor - 1 \right) + 2^n \left(\left\lfloor \frac{(x_1 - 2x_2 + x_3 + 2^{n+1}) + z_0{}^* 2^n}{2} \right\rfloor - 1 \right) \right\} \right|_{(2^{2n}-1)} \\ &= \left| \left\{ \left(\left\lfloor \frac{(x_1 - x_3) + z_0{}^* 2^n + 2^{n+1} - 2}{2} \right\rfloor \right) + 2^n \left(\left\lfloor \frac{(x_1 - 2x_2 + x_3 + 2^{n+1}) + z_0{}^* 2^n - 2}{2} \right\rfloor \right) \right\} \right|_{(2^{2n}-1)} \\ &= \left| \left\{ \left(\left\lfloor \frac{(x_1 + z_0{}^* 2^n) + (2^n - 1 - x_3) + (2^n - 1)}{2} \right\rfloor \right) + 2^n \left(\left\lfloor \frac{(x_1 + z_0{}^* 2^n) + x_3 + 2(2^n - 1 - x_2)}{2} \right\rfloor \right) \right\} \right|_{(2^{2n}-1)} \end{aligned}$$

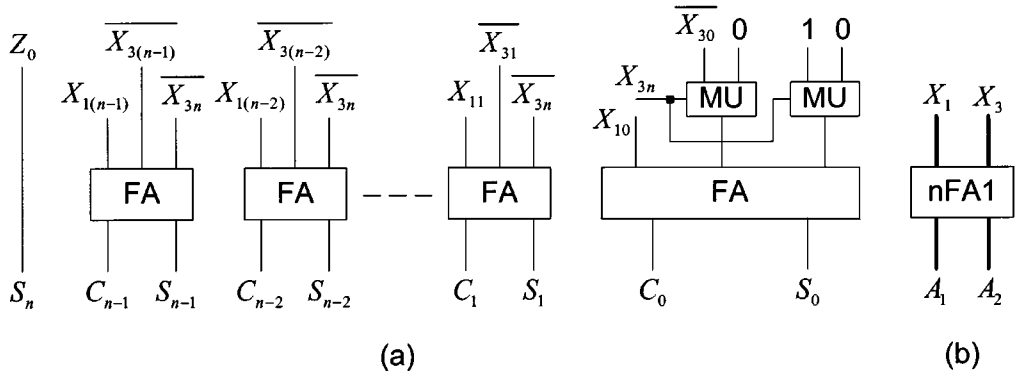

Fig. 1. Compute A using n FAs.

IV. New Converters

In Section III, we presented the necessary formulas for residue to binary conversion. In this section, we propose new converters using $2n$-bit or n-bit adders based on the formulas (8.1)–(8.4).

A. Basic Operations to Compute A and B

We have to compute the numbers $(x_1 + z_0{}^*2^n) + (2^n - 1 - x_3) + (2^n - 1)$ and $(x_1 + z_0{}^*2^n) + x_3 + 2(2^n - 1 - x_2)$ in order to obtain the values of

$$A = \left\lfloor \frac{(x_1 + z_0{}^*2^n) + (2^n - 1 - x_3) + (2^n - 1)}{2} \right\rfloor \text{ and}$$

$$B = \left\lfloor \frac{(x_1 + z_0{}^*2^n) + x_3 + 2(2^n - 1 - x_2)}{2} \right\rfloor.$$

Let $t = (2^n - 1 - x_3) + (2^n - 1)$.

If $x_3 = 2^n$, then $x_{3n} = 1$, $x_{3(n-1)} = \cdots = x_{31} = x_{30} = 0$; $t = (2^n - 2)$, which implies that $t_{n-1} = \cdots = t_1 = 1 = \bar{x}_{3(n-1)} = \cdots = \bar{x}_{31}$, $t_0 = 0$. Therefore, we have

$$t = t_{n-1} \cdots t_1 t_0 + 0 = \bar{x}_{3(n-1)} \cdots \bar{x}_{31} 0 + \bar{x}_{3n} \cdots \bar{x}_{3n} 0. \quad (9)$$

If $x_3 < 2^n$, i.e., $x_{3n} = 0$, we have

$$\begin{aligned} t &= (2^n - 1 - x_3) + (2^n - 1) \\ &= \bar{x}_{3(n-1)} \cdots \bar{x}_{31} \bar{x}_{30} + \underbrace{1 \cdots 1}_{n} \\ &= \bar{x}_{3(n-1)} \cdots \bar{x}_{31} 0 + \underbrace{\bar{x}_{3n} \cdots \bar{x}_{3n}}_{n-1} 0 + (\bar{x}_{30} + 1) \end{aligned}$$

i.e., we have

$$\begin{aligned} t &= (2^n - 1 - x_3) + (2^n - 1) \\ &= \bar{x}_{3(n-1)} \cdots \bar{x}_{31} 0 + \underbrace{\bar{x}_{3n} \cdots \bar{x}_{3n}}_{n-1} 0 + (\bar{x}_{30} + 1). \end{aligned} \quad (10)$$

Therefore, we have

`if` $x_{3n} = 1$

$$\begin{aligned} &(x_1 + z_0{}^*2^n) + (2^n - 1 - x_3) + (2^n - 1) \\ &\quad = (x_1 + z_0{}^*2^n) + \bar{x}_{3(n-1)} \cdots \bar{x}_{31} 0 + \bar{x}_{3n} \cdots \bar{x}_{3n} 0 \end{aligned}$$

`else`

$$\begin{aligned} &(x_1 + z_0{}^*2^n) + (2^n - 1 - x_3) + (2^n - 1) \\ &\quad = (x_1 + z_0{}^*2^n) + \bar{x}_{3(n-1)} \cdots \bar{x}_{31} 0 \\ &\qquad + \bar{x}_{3n} \cdots \bar{x}_{3n} 0 + (\bar{x}_{30} + 1). \end{aligned}$$

The addition of $(x_1 + z_0{}^*2^n) + (2^n - 1 - x_3) + (2^n - 1)$ is shown in Fig. 1(a) and (b). Fig. 1(b) shows the block diagram of the unit. It consists of n FAs, two MUXs, one XOR gate, and $n + 1$ inverters. The delay of this unit is the delay of the FA plus the delay of an inverter and the delay of a MUX. The circuit produces two numbers $S_n S_{n-1} S_{n-2} \cdots S_1 S_0$ and $C_{n-1} C_{n-2} \cdots C_1 C_0 0$. We denote $A_1 = S_n S_{n-1} S_{n-2} \cdots S_1$ and $A_2 = C_{n-1} C_{n-2} \cdots C_1 C_0$, and then, $A_1 + A_2 = A$.

$$z_0 = x_{10} \oplus x_{30} = 1$$

$$\begin{aligned} &(x_1 + z_0{}^*2^n) + (2^n - 1 - x_3) + (2^n - 1) \\ &\quad = 1001 + 101 + 111 = 10\,101 = 21 \end{aligned}$$

$$A = \left\lfloor \frac{(x_1 + z_0{}^*2^n) + (2^n - 1 - x_3) + (2^n - 1)}{2} \right\rfloor = 1010 = 10$$

$$\begin{aligned} &(x_1 + z_0{}^*2^n) + x_3 + 2(2^n - 1 - x_2) \\ &\quad = 1001 + 0010 + 0 = 1011 = 11 \end{aligned}$$

$$B = \left\lfloor \frac{(x_1 + z_0{}^*2^n) + x_3 + 2(2^n - 1 - x_2)}{2} \right\rfloor = 101 = 5$$

$$Y = \{10 + 8^*5\}_{2^{2^*3}-1} = 2 + 8^*6 = 50$$

$$X = 7 + 8^*Y = 407.$$

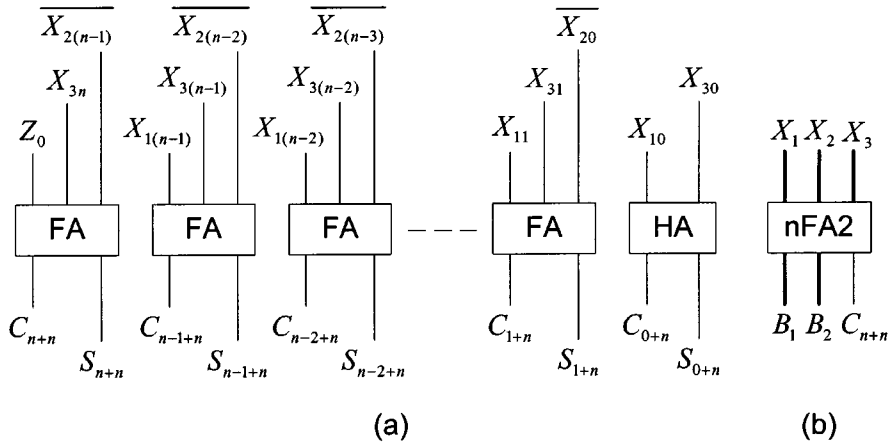

Fig. 2. Compute B using n FAs.

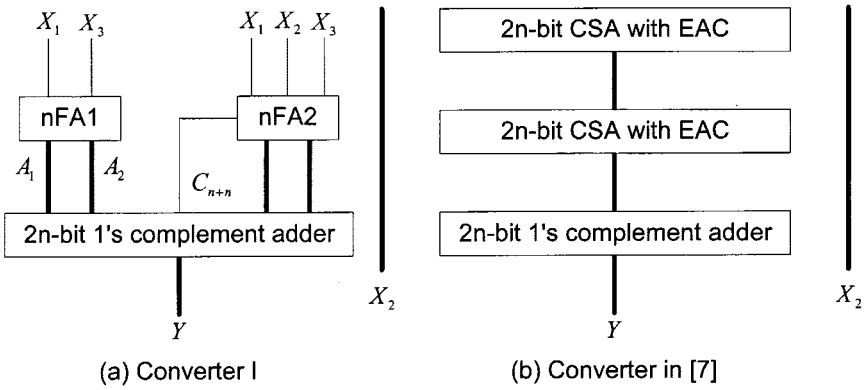

Fig. 3. $2n$-bit adder-based converters.

Next, we perform the addition $(x_1 + z_0{}^*2^n) + x_3 + 2(2^n - 1 - x_2)$ using FAs. The signal z_0 is connected to the carry-in bit of the full adder at the last FA in Fig. 2(a). Fig. 2(b) shows the block diagram of the unit. It consists of n inverters, one HA, and n FAs. The delay of this unit (nFA2) is the delay of a full adder plus the delay of an inverter. The circuit produces two numbers $S_{n+n}S_{n-1+n}S_{n-2+n}\cdots S_{1+n}S_{0+n}$, $C_{n+n}C_{n-1+n}C_{n-2+n}\cdots C_{1+n}C_{0+n}$. We denote $B_1 = S_{n+n}S_{n-1+n}S_{n-2+n}\cdots S_{1+n}$ and $B_2 = C_{n-1+n}C_{n-2+n}\cdots C_{1+n}C_{0+n}$, and then, $B_1 + B_2 + C_{n+n}{}^*2^n = B$.

Therefore, Y, as defined in (8.2), now becomes

$$\begin{aligned} Y &= |A + 2^{n*}B|_{2^{2n}-1} \\ &= |(A_1 + A_2) + 2^{n*}(B_1 + B_2 + C_{n+n}{}^*2^n)|_{2^{2n}-1} \\ &= |(A_1 + A_2 + C_{n+n}) + 2^{n*}(B_1 + B_2)|_{2^{2n}-1} \end{aligned}$$

i.e.,

$$Y = |(A_1 + A_2 + C_{n+n}) + 2^{n*}(B_1 + B_2)|_{2^{2n}-1} \qquad (11)$$

where A_1, A_2, B_1, and B_2 are all n-bit numbers; C_{n+n} is a one-bit number.

The addition in (11) can be done in many different ways using $2n$-bit or n-bit adders. These different implementations will be shown in the following.

B. $2n$-Bit Adder-Based Converter—Converter I

In the following, we present the new Converter I implementing the addition in (11) using a $2n$-bit adder.

$$\begin{aligned} Y &= |(A_1 + A_2 + C_{n+n}) + 2^{n*}(B_1 + B_2)|_{2^{2n}-1} \\ &= |C_{n+n} + (A_1 + 2^nB_1) + (A_2 + 2^nB_2)|_{2^{2n}-1} \\ &= |C_{n+n} + S_{n+n}S_{n-1+n}S_{n-2+n}\cdots \\ &\qquad S_{1+n}S_nS_{n-1}S_{n-2}\cdots S_1 + C_{n-1+n}C_{n-2+n}\cdots \\ &\qquad C_{1+n}C_{0+n}C_{n-1}C_{n-2}\cdots C_1C_0|_{(2^{2n}-1)} \end{aligned}$$

where $S_{n+n}S_{n-1+n}S_{n-2+n}\cdots S_{1+n}S_nS_{n-1}S_{n-2}\cdots S_1$, and $C_{n-1+n}C_{n-2+n}\cdots C_{1+n}C_{0+n}C_{n-1}C_{n-2}\cdots C_1C_0$ are two $2n$-bit numbers, and C_{n+n} is a one-bit number.

In Fig. 3(a), the units nFA1 and nFA2, which are used to produce A_1, A_2, B_1, and B_2, are connected to a $2n$-bit 1's complement adder. The $2n$-bit adder produces the value Y, which forms the $2n$ MSBs of the number X, whereas x_2 forms the n LSBs of X.

The hardware required in the new Converter I shown in Fig. 3(a) is as follows: $2n$ FAs, one HA, two MUXs, one XOR gate, $2n + 1$ inverters, and one $2n$-bit 1s complement adder. The delay of the converter t_{conv} is the sum of the delay of the FA t_{FA}, the delay of an inverter t_{inv}, the delay of MUX t_{MUX}, and the delay of the $2n$-bit

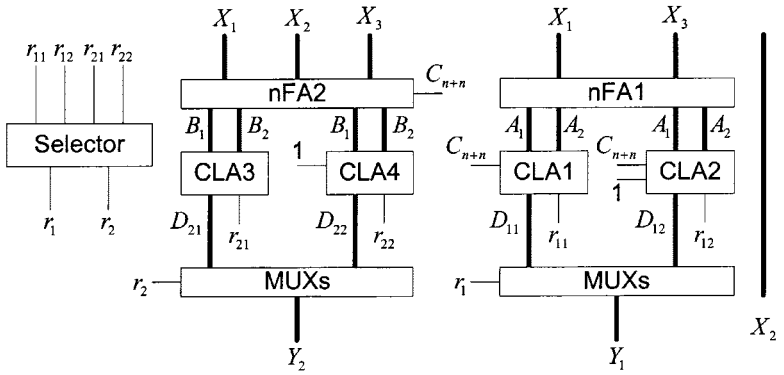

Fig. 4. Converter II—using four n-bit adders.

1s complement adder $t_{1CA(2n)} = 2t_{CPA(2n)}$ [7], i.e., $t_{conv} = t_{FA} + t_{inv} + 2t_{CPA(2n)} + t_{MUX}$.

In the literature, one of the best converters using $2n$-bit adders is presented in [7]. In order to compare the performance, we show the main components used in the converter proposed in [7] as Fig. 3(b). The delay in [7] is $2t_{FA} + t_{inv} + 2t_{CPA(2n)} + 3t_{MUX}$. For simplicity reasons, we only compare one version of the implementation in [7]. The second implementation has the same result. From the side-by-side comparison, it is easy to see that we save one $2n$-bit CSA with end around carry (EAC).

Detailed comparison of the other related converters are summarized in Table I, where the data for [8], [9], and [11] are from [7, Table I]. In summary, Converter I is the best converter using $2n$-bit adders, using about half of the hardware used in [7]. The reason for such improvement is that the converters in [8], [9], [11], and [18] use the formula $|A + B + C - Z|_{2^{2n}-1}$, where A, B, C, and Z are $2n$ bit numbers obtained from (x_1, x_2, x_3), whereas the new Converter I is derived based on the New Chinese Remainder Theorem I and is computed by $Y = |(A_1 + A_2 + C_{n+n}) + 2^{n*}(B_1 + B_2)|_{2^{2n}-1}$, which reduces the four-number operation into two numbers.

C. n-Bit Adder Based Converters—Converter II and III

The addition in (11) can also be done by n-bit adders. In this section, we propose two such converters. The performance is to be compared with the performance of the converter in [6], [15], and [18], which use n-bit adders as well. Since we can only generate n-bit numbers using n-bit adders, we therefore obtain the value Y in the form $Y = Y_1 + 2^{n*}Y_2$, where Y_1 and Y_2 are both n-bit binary numbers.

Recall that $Y = |(A_1 + A_2 + C_{n+n}) + 2^{n*}(B_1 + B_2)|_{2^{2n}-1}$, where A_1, A_2, B_1, and B_2 are all n-bit numbers; C_{n+n} is a one bit number. Using an n-bit adder, we can add A_1 and A_2 together with C_{n+n}, which generates a sum D_1 and a carry r_1. Similarly, we can add B_1 and B_2 using an n-bit adder, which generates a sum D_2 and a carry r_2. Since the addition is module $2^{2n} - 1$ addition, the carry r_2 represents a number that should be added to the number $A_1 + A_2 + C_{n+n}$. For the case where the carry r_2 is 0, the sum D_1 is the value Y_1. For the case where the carry r_1 is 0, the sum D_2 is the value Y_2 such that $Y = Y_1 + 2^{n*}Y_2$. However, when the carries r_1 and r_2 are not 0, the value D_1 and D_2 must be modified to obtain the correct value of Y_1 and Y_2. In the following, we propose Converter II and III for the operation. Compared with Converter II, Converter III achieves faster speed while using more hardware.

Converter II: In Fig. 4, we use two carry look ahead (CLA) adders to perform the operation $A_1 + A_2$ and $A_1 + A_2 + 1$ in parallel. The results are denoted as D_{11} and D_{12} with carry r_{11} and r_{12}, respectively. If $r_{11} \neq r_{12}$, we have $D_{11} = 2^n - 1$ and $D_{12} = D_{11} + 1 = 2^n$. Similarly, two CLAs are used to perform $B_1 + B_2$ and $B_1 + B_2 + 1$, whereas the results are denoted as D_{21} and D_{22} with carry r_{21} and r_{22}. If $r_{21} \neq r_{22}$, we have $D_{21} = 2^n - 1$ and $D_{22} = D_{21} + 1 = 2^n$.

The selector module selects the correct carry and the correct sum for the number Y_1 and Y_2. The function of the selector is described in the following.

```
If r11 ≠ r12 and r21 ≠ r22, then r1 = r2 = 0.
Else if r11 = r12, then r1 = r11.
   If r1 = 0, r2 = r21
   else r2 = r22.
Else if r21 = r22, then r2 = r21.
   If r2 = 0, r1 = r11
   else r1 = r12.
```

Therefore, the carry $r_1 = 1$ if $(r_{11} = r_{12} = 1)$ or $(r_{21} = r_{22} = 0$ and $r_{11} = 1)$ or $(r_{21} = r_{22} = 1$ and $r_{12} = 1)$, i.e., $r_1 = r_{11}r_{12} + \bar{r}_{21}\bar{r}_{22}r_{11} + r_{21}r_{22}r_{12}$. Similarly, $r_2 = r_{21}r_{22} + \bar{r}_{11}\bar{r}_{12}r_{21} + r_{11}r_{12}r_{22}$.

The selector implements these two functions. Note here that the selector does not introduce any extra delay since CLAs are used, and the carries r_{11}, r_{12}, r_{21}, and r_{22} are generated during the carry-generation phase of the CLAs and are available for evaluation to the selector while the CLAs perform the summation.

The hardware required in Fig. 4 includes $2n$ FAs, one HA, $2 + 2n$ MUXs, one XOR gate, $2n + 5$ inverters (including four inverters for the selector), two AND gates for the selector, and $4n$-bit CLAs. The delay of the converter is $t_{conv} = t_{inv} + t_{FA} + t_{CLA(n)} + t_{MUX}$.

Converter III: Considering the fact that $D_{12} = D_{11} + 1$ and $D_{22} = D_{21} + 1$, we can replace the CLA2 and CLA4 in Fig. 4 by other combinational circuits that perform the operation $D_{12} = D_{11} + 1$ and $D_{22} = D_{21} + 1$. Fig. 5 shows such a converter.

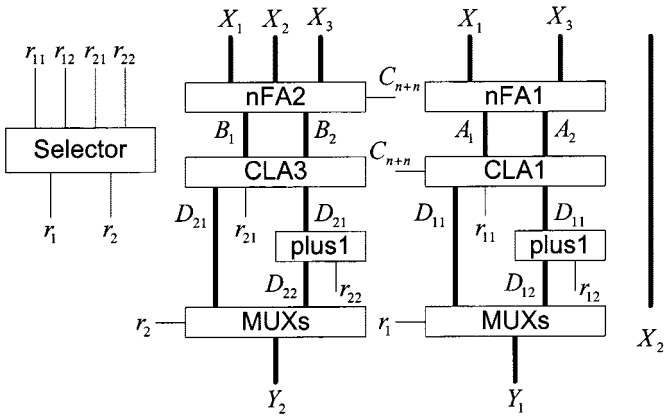

Fig. 5. Converter III—using 2 n-bit adders.

The circuit *plus1* performs the function of adding 1 to a n-bit input number. Consider $D = d_{n-1}d_{n-2}\cdots d_1d_0$, $D+1 = d_{n-1}d_{n-2}\cdots d_1d_0+1 = e_ne_{n-1}e_{n-2}\cdots e_1e_0$. We have the following equations, which imply that the circuit *plus1* requires $n-1$ XOR gates and n AND gates plus 1 inverter.

$$e_0 = \bar{d}_0; \quad e_1 = d_1 \oplus d_1d_0; \quad e_i = d_i \oplus d_{i-1}\cdots d_0;$$
$$e_{n-1} = d_{n-1} \oplus d_{n-2}\cdots d_0; \quad e_n = d_{n-1}d_{n-2}\cdots d_0.$$

The hardware required in Fig. 5 includes $2n$ FAs, $2+2n$ MUXs, $1+2(n-1)$ XOR gate, $2n+5+2$ inverters (including four inverters for the selector and two for the plus-1 circuit), $2+2n$ AND gates for the selector and the *plus-1* circuit, one HA, and $2n$-bit CLAs. The delay of the converter is $t_{conv} = t_{inv}+t_{FA}+t_{XOR}+t_{CLA(n)}+t_{AND}+t_{MUX}$.

In order to make clear comparison, the Fig. 6 shows the main components for the converter proposed in [6]. No detailed implementation is given for each module in [6]. We evaluate the performance based on [4]. Recently, the results in [4] are also used to evaluate the performance in [7]. Modules M1 and M2 require two CLAs and one CSA, where all are n-bit adders, one XOR for generating C1, and $2n$ inverters for 2s complement operation. M3 and M4 require two additional CPAs and $2n$ inverters for 2s complement opseration. Module M6 uses nine AND gates, one OR gate, eight inverters, and one XOR gate. M5 uses 8^*n bit memory to store the value. The delay is $t > (2t_{inv}+2t_{CPA(n)}+t_{FA})+t_{XOR}+t_{inv}+t_{AND} = 3t_{inv}+t_{FA}+t_{XOR}+t_{AND}+2t_{CPA(n)}$.

Two more recent converters using n-bit adders have also been proposed in [15] and [18]. The one in [18] is based on the approach in [7] and, therefore, has high hardware cost, whereas the one in [15] has similar hardware cost as the new converters proposed here. However, the converter in [15] has some unused dynamic range.

Table II summarizes the comparison of the two converters proposed in this paper as well as the converter in [6], [15], and [18], where CII stands for Converter II, whereas CIII stands for Converter III.

Assume $t_{MUX} = 2$, $t_{FA} = 2$, $t_{inv} = 1 = t_{AND}$, $t_{CLA(n)} = \log n$, $t_{XOR} = 2$; then the delay of Converter II is $t_{inv}+t_{FA}+t_{MUX}+t_{CLA(n)} = 5+\log n$. The delay of Converter III is $t_{inv}+t_{FA}+t_{XOR}+t_{AND}+t_{MUX}+t_{CLA(n)} = 8+\log n$. The delay of the converter in [6] is

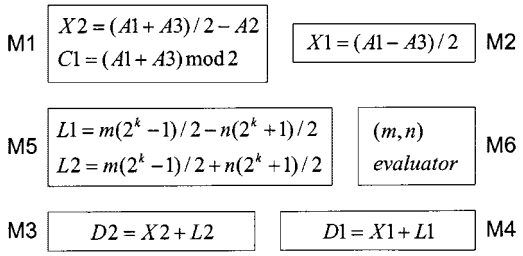

Fig. 6. Converter proposed in [6].

$3t_{inv}+t_{FA}+t_{XOR}+t_{AND}+2t_{CLA(n)} = 8+2\log n$. The delay of Converter II is almost half of the delay of the converter in [6].

Assume the straightforward implementation of the CLA, which consists of a carry look-ahead unit and a summation unit, which, in total, require $2n+(1/2)n(n+1)$ AND gates, $2n$ XOR gates, and n OR gates. The hardware requirement in [6] is even higher than the hardware required in Converter III, whereas its delay is longer.

V. Conclusion

Three different residue-to-binary converters for the special moduli $(2^n-1, 2^n, 2^n+1)$ have been presented in this paper. Compared with various previous proposed converters, the new converters proposed here have better performance in terms of speed and area. The new converters are designed based on the recently introduced New Chinese Remainder Theorems. It is expected that for other moduli sets, the New Chinese Remainder Theorems will also improve the design of residue-to-binary converters.

Acknowledgment

The authors kindly acknowledge detailed comments from referees, which have improved the quality of the manuscript.

References

[1] H. L. Garner, "The residue number system," *IRE Trans. Electron. Comput.*, vol. EC-8, pp. 140–147, June 1959.
[2] N. Szabo and R. Tanaka, *Residue Arithmetic and its Applications to Computer Technology*. New York: McGraw-Hill, 1967.
[3] M. A. Soderstrand *et al.*, Ed., *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*. New York: IEEE, 1986.
[4] Y. Hwang, *Computer Arithmetic Principles, Architecture, and Design*. New York: Wiley, 1979.
[5] A. Ashur, M. K. Ibrahim, and A. Aggoun, "Novel RNS structures for the moduli set $(2^n-1, 2^n, 2^n+1)$ and their application to digital filter implementation," *Signal Process.*, vol. 46, pp. 331–343, 1995.
[6] D. Gallaher, F. Petry, and P. Srinivasan, "The digital parallel method for fast RNS to weighted number system conversion for specific moduli $(2^k-1, 2^k, 2^k+1)$," *IEEE Trans. Circuits Syst. II*, vol. 44, pp. 53–57, Jan. 1997.
[7] S. Piestrak, "A high-speed realization of a residue to binary number system converter," *IEEE Trans. Circuits Syst. II*, vol. 42, Oct. 1995.
[8] K. Ibrahim and S. Saloum, "An efficient residue to binary converter design," *IEEE Trans. Circuits Syst.*, vol. 35, pp. 1156–1158, Sept. 1988.
[9] S. Andraos and H. Ahmad, "A new efficient memoryless residue to binary converter," *IEEE Trans. Circuits Syst.*, vol. 35, pp. 1441–1444, Nov. 1988.
[10] F. Taylor and A. S. Ramnarynan, "An efficient residue-to-decimal converter," *IEEE Trans. Circuits Syst.*, vol. CAS-28, Dec. 1981.
[11] A. Dhurkadas, "An efficient residue to binary converter design," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 849–850, June 1990.

[12] Y. Wang and M. Abd-el-Barr, "A new algorithm for RNS decoding," *IEEE Trans. Circuits Syst. I*, vol. 43, pp. 998–1001, Dec. 1996.
[13] Y. Wang, "Residue-to-binary converters based on new chinese remainder theorems," *IEEE Trans. Circuits Syst. II*, pp. 197–206, Mar. 2000.
[14] ——, "New Chinese remainder theorems," *Proc. 32th Asilomar Conf. Signals, Syst., Comput.*, vol. 1, pp. 165–171, 1998.
[15] R. Conway and J. Nelson, "Fast converter for 3 moduli RNS using new property of CRT," *IEEE Trans. Comput.*, vol. 48, pp. 852–860, Aug. 1999.
[16] B. Vinnakota and V. V. B. Rao, "Fast conversion techniques for binary-residue number systems," *IEEE Trans. Circuits Syst. I*, vol. 41, pp. 927–929, Dec. 1994.
[17] W. J. Jenkins, "Techniques for residue-to-analog conversion for residue-encoded digital filters," *IEEE Trans. Circuits Syst.*, vol. CAS-25, pp. 555–562, July 1978.
[18] M. Bhardwaj, A. B. Premkumar, and T. Srikanthan, "Breaking the $2n$-bit carry propagation barrier in residue to binary conversion for the $(2^n - 1, 2^n, 2^n + 1)$ module set," *IEEE Trans. Circuits Syst. II*, vol. 45, pp. 998–1002, Sept. 1998.

Yuke Wang received the B.Sc. degree from the University of Science and Technology of China, Hefei, in 1989 and the M.Sc. and Ph.D. degrees from the University of Saskatchewan, Saskatoon, SK, Canada, in 1992 and 1996, respectively.

He has held faculty positions at Concordia University, Montreal, QC, Canada, and Florida Atlantic University, Boca Raton. Currently, he is an Assistant Professor with the Computer Science Department, University of Texas at Dallas, Richardson. He has also held Visiting Assistant Professor positions with the University of Minnesota, Minneapolis, the University of Maryland, College Park, and the University of California, Berkeley. His research interests include VLSI design of circuits and systems for DSP and communications, computer-aided design, and computer architectures. From 1996 to 2001, he has published about 60 papers, among which, about 20 papers that have appeared in IEEE/ACM Transactions. He is an Editor of *Applied Signal Processing*.

Dr. Wang is currently an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, PART II and of the IEEE TRANSACTIONS ON VLSI SYSTEMS.

Xiaoyu Song received the M.S. and Ph.D. degrees in computer engineering from University of Pisa, Pisa, Italy, in 1987 and 1992, respectively.

From 1992 to 1997, he was a Faculty Member with the University of Montreal, Montreal, QC, Canada. He was a senior member of consulting staff at Cadence, San Jose, CA. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Portland State University, Portland, OR. His research interests include IC and VLSI circuit design, testing and verification, systems on a chip, and synthesis. He serves on the Editorial Board of *VLSI Design: An International Journal of Custom-Chip Design, Simulation, and Testing*.

Dr. Song is an editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS and the IEEE TRANSACTIONS ON VLSI SYSTEMS. He has served on many program committees, such as the IEEE International Conference on Quality of Electronics and the ACM International Conference on System-Level Interconnect Prediction.

Mostapha Aboulhamid (M'82) received the Ph.D. degree from the University of Montreal, Montreal, QC, Canada, in 1985 and the engineering degree from the University of Grenoble, Grenoble, France, in 1974.

He is currently a Professor at the University of Montreal. His research interests are in hardware/software modeling and synthesis, paradigms for design reuse, and hardware description languages.

Hong Shen received the B.Eng. degree from Beijing University of Science and Technology, Beijing, China, the M.Eng. degree from the University of Science and Technology of China, Hefei, and the Ph.Lic. and Ph.D. degrees from Abo Academi University, Abo, Finland, all in computer science.

He is currently a Professor of computer science with the Graduate School of Information Science, Japan Advanced Institute of Science and Technology, Tatsunokuchi, Ishikawa. Previously, he was a Professor with Griffith University, Brisbane, Australia. He has published over 140 technical papers on algorithms, parallel and distributed computing, interconnection networks, parallel databases and data mining, multimedia systems, and networking. He has served as an editor of *Parallel and Distributed Computing Practice*, Associate Editor of the *International Journal of Parallel and Distributed Systems and Networks*, editorial board member of *Parallel Algorithms and Applications*, the *International Journal of Computer Mathematics*, and the *Journal of Supercomputing*.

Dr. Shen has been the chair/committee member of various international conferences.