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

Yuke Wang
Department of Electrical and Computer Engineering, Concordia University
Xiaoyu Song, Mostapha Aboulhamid
Departement d'informatique et recherche operationnelle , Universite de Montreal
Montreal, Quebec

Abstract

This paper proposes three new residue-to-binary converters using 2n- bit or n-bit adders for the three moduli residue number system of the form $(2^n-1, 2^n, 2^n+1)$. *The 2n- bit adder based converter is faster and requires about half of the hardware required by previous methods. For n-bit adder based implementations, one new converter is twice as fast as the previous method using similar amount of hardware; while another new converter achieves improvement in both speed and area.*

Keywords residue number system, arithmetic, circuit, algorithm, adders.

I Introduction

There has been interest in Residue Number Systems arithmetic as a basis for computational hardware since the 1950's [1] [2]. During the past decade, the residue number system (RNS) has received a 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 wide spread use are the inherent properties of RNS such as parallelism, modularity, fault tolerance and carry free operations [3]. The conversion from binary to residue and vice versa is the crucial step for any successful RNS application. In recent years, the conversion process has been studied very intensively [5-12]. For general moduli sets, the residue to binary conversions are mainly based on the Chinese Remainder Theorem or Mixed-Radix Conversion.

Due to relatively simple conversion, 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]. Several conversion methods for $(2^n-1, 2^n, 2^n+1)$ have been reported [6] [7] [8] [9] [10] [11]. The method proposed in [6] is the first one to use n- bit CPAs, where multiplication, division, and table look-up are also needed. The approaches in [7] [8] [9] using FAs and $2n$-bit CPAs. Among them, [7] has the best implementation using $2n$-bit adders.

In this paper, we present three different converters using either $2n$- bit or n- bit adders. The $2n$-bit adder based converter is faster and requires about half of the hardware required by previous methods [7][8][9]. For n-bit adder based implementations, one new converter is twice as fast as the previous method [6] using similar amount of hardware; while another new converter achieves improvement in both speed and area.

In the following, we first present the new conversion formulas; then we show an example and propose three different hardware implementations. Due to limited space, formulas introduced are without proof. Detailed proofs can be found in [13].

II Mathematical 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}. A residue number system is defined in terms of a set of relatively prime moduli set $(P_1, P_2, \ldots, P_k)$, where $GCD(P_i, P_j) = 1$ for $i \ne j$. A binary number X can be represented as $X = (x_1, x_2, \ldots, x_k)$, where $x_i = X \bmod P_i$. The representation is unique for any $X \in [0, M-1]$, $M = \prod_{1 \le i \le k} P_i$.

If $(P_1, P_2, P_3) = (2^n-1, 2^n, 2^n+1)$, X can be represented by a tuple (x_1, x_2, x_3), where $x_1 = x_{1(n-1)}x_{1(n-2)}\ldots x_{11}x_{10}$ and $x_2 = x_{2(n-1)}x_{2(n-2)}\ldots x_{21}x_{20}$ are two n-bit binary numbers; $x_3 = x_{3n}x_{3(n-1)}x_{3(n-2)}\ldots x_{31}x_{30}$ is an $n+1$-bit binary number. The RNS to binary converter computes the number X from the tuple (x_1, x_2, x_3).

Theorem [13] The number X can be computed from (x_1, x_2, x_3) by the formula:

0-8186-8409-7/98 $10.00 © 1998 IEEE

$X = x_2 + 2^n * [(x_2 - x_3) + (x_1 - 2x_2 + x_3)2^{n-1}(2^n + 1)]_{(2^{2n}-1)}$

which can be further processed as

$X = x_2 + 2^n * Y$ (1)

$Y = \{A + 2^n * B\}_{(2^{2n}-1)}$ (2)

$$A = \left\lfloor \frac{(x_1 + (x_{10} \oplus x_{30}) * 2^n) + (2^n - 1 - x_3) + (2^n - 1)}{2} \right\rfloor \quad (3)$$

$$B = \left\lfloor \frac{(x_1 + (x_{10} \oplus x_{30}) * 2^n) + x_3 + 2(2^n - 1 - x_2)}{2} \right\rfloor \quad (4)$$

Example Consider the example shown in [6]. Let $(2^n - 1, 2^n, 2^n + 1) = (7, 8, 9)$ and an octal number 627, which can be represented as (1, 7, 2)=(001, 111, 0010). Compared with the long calculation in page 56 in [6], the following process is much simpler.

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

$(x_1 + (x_{10} \oplus x_{30}) * 2^n) + (2^n - 1 - x_3) + (2^n - 1)$

$= 1001 + 101 + 111 = 10101$

$$A = \left\lfloor \frac{(x_1 + (x_{10} \oplus x_{30}) * 2^n) + (2^n - 1 - x_3) + (2^n - 1)}{2} \right\rfloor = 1010$$

$(x_1 + (x_{10} \oplus x_{30}) * 2^n) + x_3 + 2(2^n - 1 - x_2) =$

$1001 + 0010 + 0 = 1011$

$$B = \left\lfloor \frac{(x_1 + (x_{10} \oplus x_{30}) * 2^n) + x_3 + 2(2^n - 1 - x_2)}{2} \right\rfloor = 101$$

$Y = \{1010 + 8 * 101\}_{2^{2*3}-1} = 2 + 8 * 6$

$X = 7 + 8 * Y = 7 + 2 * 8 + 6 * 8^2$

III New Converters

In this section, we propose new converters using *2n*-bit or *n*-bit adders based on formulas (1), (2), (3) and (4).

(1) Basic Operations to Compute *A* and *B*

If $x_3 = 2^n$, then $x_{3n} = 1$, $x_{3(n-1)} = \ldots = x_{31} = x_{30} = 0$;

$(2^n - 1 - x_3) + (2^n - 1) = (2^n - 2)$

$= \bar{x}_{3(n-1)} \ldots \bar{x}_{31} 0 + \bar{x}_{3n} \ldots \bar{x}_{3n} 0$.

If $x_3 < 2^n$, i.e., $x_{3n} = 0$, then $(2^n - 1 - x_3) + (2^n - 1) = \bar{x}_{3(n-1)} \ldots \bar{x}_{31} \bar{x}_{30} + 1 \ldots 1$

$= \bar{x}_{3(n-1)} \ldots \bar{x}_{31} 0 + \bar{x}_{3n} \ldots \bar{x}_{3n} 0 + (\bar{x}_{30} + 1)$.

The addition of $(x_1 + z_0 * 2^n) + (2^n - 1 - x_3) + (2^n - 1)$ is shown in Figure 1(a) and (b). Figure 1(b) shows the block diagram of the unit. The circuit produces two numbers $S_n S_{n-1} S_{n-2} \ldots S_1 S_0$ and $C_{n-1} C_{n-2} \ldots C_1 C_0 0$. We denote $A_1 = S_n S_{n-1} S_{n-2} \ldots S_1$ and $A_2 = C_{n-1} C_{n-2} \ldots C_1 C_0$, then $A_1 + A_2 = A$.

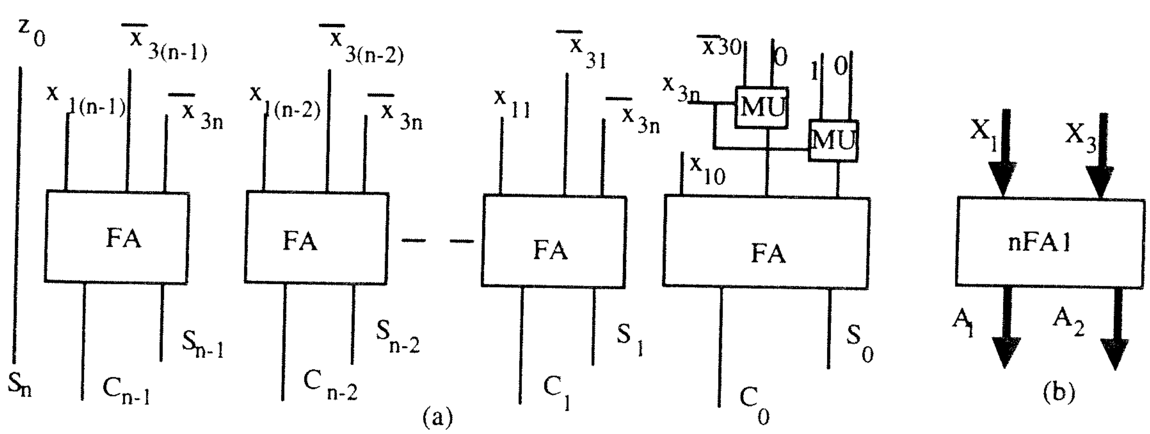

Figure 1 Compute *A* Using *n* FAs

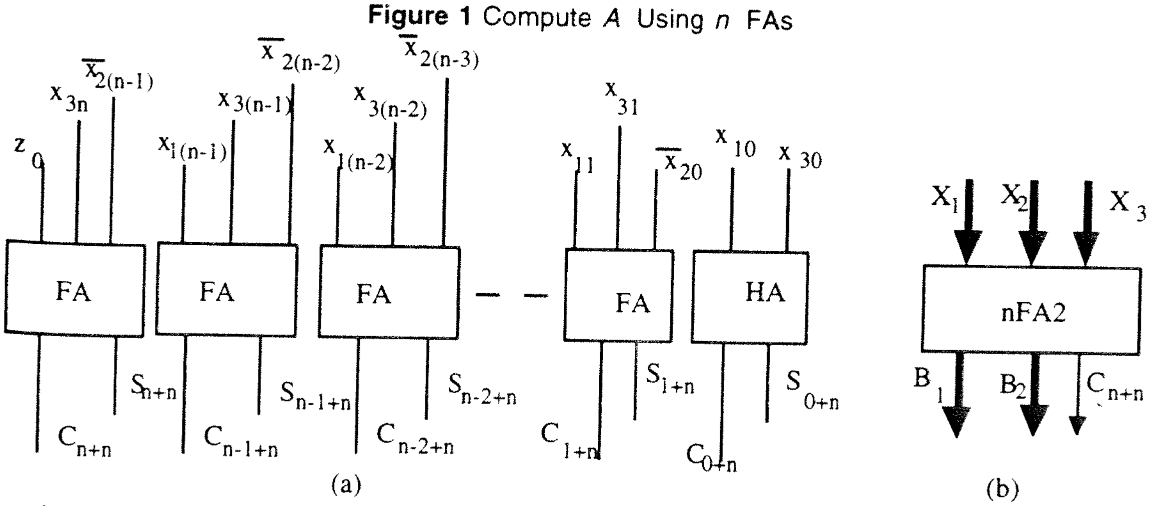

Figure 2 Compute *B* Using *n* FAs

Figure 2 (b) shows the block diagram of the unit performing $(x_1 + z_0 * 2^n) + x_3 \quad +2(2^n - 1 - x_2)$ using FAs. The circuit produces two numbers $S_{n+n}S_{n-1+n}S_{n-2+n}...S_{1+n}S_{0+n}$, $C_{n+n}C_{n-1+n}C_{n-2+n}...C_{1+n}C_{0+n}$. Let $B_1 + B_2 + C_{n+n} * 2^n = B$, where we have $B_1 = S_{n+n}S_{n-1+n}S_{n-2+n}...S_{1+n}$ $B_2 = C_{n-1+n}C_{n-2+n}...C_{1+n}C_{0+n}$.

Therefore we have the formula

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

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

i.e., $$Y = \{(A_1 + A_2 + C_{n+n}) + 2^n * (B_1 + B_2)\}_{2^{2n}-1} \qquad (5)$$

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

The addition in (5) can be done in many different ways using $2n$- bit or n- bit adders. These different implementations will be shown below.

(2) *2n*-bit Adder Based Converter - Converter I

Next we show the Converter I which implements formula (5) using a $2n$- bit adder.

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

$$= \{C_{n+n} + (A_1 + 2^n B_1) + (A_2 + 2^n B_2)\}_{2^{2n}-1}$$

$$= \{C_{n+n} + S_{n+n}S_{n-1+n}S_{n-2+n}...S_{1+n}S_nS_{n-1}S_{n-2}...S_1$$
$$+ C_{n-1+n}C_{n-2+n}...C_{1+n}C_{0+n}C_{n-1}C_{n-2}...C_1C_0\}_{(2^{2n}-1)}$$

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

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

Figure 3(b) shows the components in the converter proposed in [7]. It is easy to see that we save one $2n$-bit CSA with EAC. Detailed comparison of the related other converters are summarized in the following Table 1, where the data for references [8] [9] [11] are from Table I in [7].

In summary, Converter I is the best converter based on $2n$-bit adders. It saves almost half of the hardware required by the previous best converter while increasing the speed.

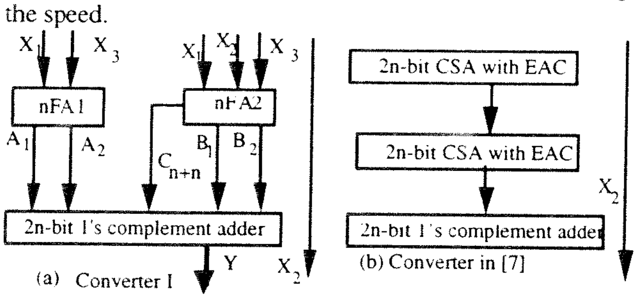

Figure 3 *2n* -bit Adder Based Converters

(3) *n*-bit Adder Based Converters - Converter II and III

The addition in formula (5) can also be done by n-bit adders, which generates the value Y in the form $Y = Y_1 + 2^n * Y_2$ such that Y_1 and Y_2 are both n- bit binary numbers.

In Figure 4, we use two carry look ahead adders (CLA) 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 $B_1 + B_2 + 1$ while 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 below.

If $r_{11} \neq r_{12}$ and $r_{21} \neq r_{22}$, then $r_1 = r_2 = 0$

Else if $r_{11} = r_{12}$, then $r_1 = r_{11}$; if $r_1 = 0$, $r_2 = r_{21}$, else $r_2 = r_{22}$

Else if $r_{21} = r_{22}$, then $r_2 = r_{21}$; if $r_2 = 0$, $r_1 = r_{11}$, $r_1 = r_{12}$

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.

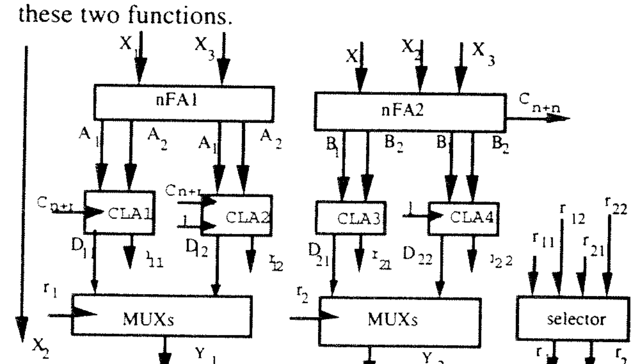

Figure 4 Converter II - Using 4 *n-bit* Adders

Considering the fact that $D_{12} = D_{11} + 1$ and $D_{22} = D_{21} + 1$, we can replace the CLA2 and CLA4 in Figure 4 by other combinational circuits that perform the operation $D_{12} = D_{11} + 1$ and $D_{22} = D_{21} + 1$. The following Figure 5 shows Converter III. The circuit *plus1* performs the function of adding 1 to an n-bit input numbers. Consider $D = d_{n-1}d_{n-2}...d_1d_0$, $D + 1 = d_{n-1}d_{n-2}...d_1d_0 + 1$ $= e_ne_{n-1}e_{n-2}...e_1e_0$. We have the following equations:

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

which imply that the circuit *plus1* requires *n-1* XOR gates and *n* AND gates plus 1 inverter.

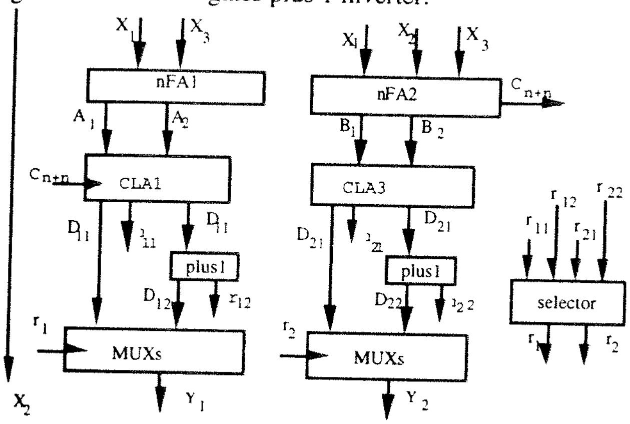

Figure 5 Converter III - Using 2 *n-bit* Adders

The following Figure 6 shows the main components for the converter proposed in [6]. No detailed implementation was given for each module in [6]. We evaluate the performance based on [4]. Modules M1 and M2 require 2 CLAs, 1 CSA, all are *n*- bit adders; 1 XOR for generating C1, *2n* inverters for 2's complement operation. M3 and M4 require two additional CPAs and *2n* inverters for 2's complement operation. Module M6 uses 9 AND gates, 1 OR gates, 8 inverters, and 1 XOR gate. M5 uses 8*n bit memory to store the value. Delay $t > (2t_{inv} + 2t_{CPA(n)} + t_{FA}) + t_{XOR} + t_{inv} + t_{AND} = 3t_{inv} + t_{FA} + t_{XOR} + t_{AND} + 2t_{CPA(n)}$. The differences in hardware and delay between Converter II (CII), Converter III (CIII) and the converter in [6] are summarize in Table 2.

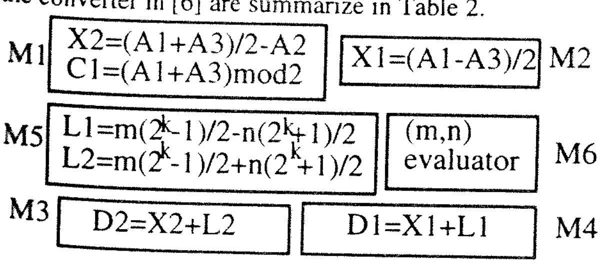

Figure 6 Converter proposed in [6]

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 $3t_{inv} + t_{FA} + t_{XOR} + t_{AND} + 2t_{CLA(n)} = 8 + 2\log n$. 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 carry look-ahead unit and a summation unit which in total require $2n + n(n+1)/2$ AND gates, *2n* XOR gates, and *n* OR gates. The hardware requirement in [6] is even higher than the hardware required in Converter III while 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. The converters can be implemented using *2n*-bit or *n*-bit adders. The *2n*- bit adder based converter is faster and requires about half of the hardware required by the previous converter in [7]. For *n*-bit adder based implementations, one new converter is twice as fast as the previous method using similar amount of hardware; while another new converter achieves improvement in both speed and area.

References

[1] H. L. Garner, "The residue number system", IRE Trans. Electronic Computers, 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., Eds, "Residue number system arithmetic: modern applications in digital signal processing", New York, IEEE Press, 1986.

[4] Hwang, "Computer arithmetic principles, architecture, and design", John Wiley and Sons. 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 processing 46 (1995) 331-343.

[6] D. Gallaher, F. Petry, and Padmini Srinivasan, " The digital parallel method for fast RNS to weighted number system conversion for specific moduli $(2^n - 1, 2^n, 2^n + 1)$", IEEE Transactions on Circuits and Systems -II, Vol. 44, No. 1, January 1997, pp. 53-57.

[7] S. Piestrak, "A high-speed realization of a residue to binary number system converter", IEEE Transactions on Circuits and Systems -II, Vol. 42, No. 10, October 1995. pp. 661-663.

[8] K. Ibrahim and S. Saloum, "An efficient residue to binary converter design", IEEE Transactions on Circuits and Systems, Vol. 35, No. 9, September 1988. pp. 1156-1158.

[9] S. Andraos and H. Ahmad, "A new efficient memoryless residue to binary converter", IEEE Transactions on Circuits and Systems , Vol. 35, No. 11, November 1988. pp. 1441-1444.

[10] F. Taylor and A. S. Ramnarynan, "An efficient residue-to-decimal converter", IEEE Transactions on Circuits and Systems, Vol. CAS-28, NO. 12, December 1981. pp. 1164-1169.
[11] A. Dhurkadas, comments on "An efficient residue to binary converter design", IEEE Transactions. on Circuits and Systems, Vol. 37, June 1990, pp. 849-850.
[12] Y. Wang and M. Abd-el-Barr, "A New Algorithm for RNS Decoding", IEEE Transactions on Circuits and Systems - I, Vol. 43, No. 12, December 1996. pp. 998-1001.
[13] Y. Wang, X. Song, and M. Aboulhamid, "Adder Based Residue to Binary Number Converters for $(2^n-1, 2^n, 2^n+1)$", Technical Report, No. 1997-1, Department of Electrical and Computer Engineering, Concordia University.

Table 1 Performance Comparison of *2n*- bit Adder Based Converters

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

Table 2 Performance Comparison of *n*- bit Adder Based Converters

	FA	MUX	XOR	AND /OR	INV	HA	Mem	CPA	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	$3t_{inv} + t_{FA} + t_{XOR} + t_{AND} + 2t_{CLA(n)}$