



VLSI Research Group

University of Windsor

*An Improved Residue to Binary
Converter*

**IEEE Transaction on Circuits and Systems (brief
contribution)**

Zhongde Wang, G. A. Jullien, and W. C. Miller

An Improved Residue to Binary Converter

Zhongde Wang, G. A. Jullien, and W. C. Miller

Abstract

An efficient algorithm, which converts residue numbers, modulo $(2^n+1, 2^n, 2^n-1)$, into an equivalent binary number, is proposed; the algorithm improves on a previously published technique. A novel hardware implementation is also discussed. In comparison with two previously published converters, the proposed converter provides at least a 40% hardware saving and a 45% reduction in the delay.

Keywords: Residue Arithmetic; Residue to Binary Converter; 3 Modulus Residue System

1.0 Introduction

The three moduli $(2^n + 1, 2^n, 2^n - 1)$ residue number system (RNS) possesses many advantages over other moduli sets. The residues are easily obtained, and the system can be efficiently scaled by any one of the chosen moduli [1]-[7]. As with all RNS arithmetic systems, the conversion from residue to binary is the most computationally intensive task. Among all published algorithms for conversion from residue to binary, the algorithm given by Andraos and Ahmed [3] is generally acknowledged to be the most efficient.

In this paper we make substantial improvements to the algorithm proposed by Andraos and Ahmed [3]. As a result, a hardware saving in excess of 40% and a reduction of 45% in the critical path is obtained.

2.0 Andraos and Ahmed's algorithm

To convert a number, X , from its residue representation (r_1, r_2, \dots, r_L) into its binary counterpart, the Chinese remainder theorem of eqn. (1) should be applied:

$$X = \left| \sum_i \hat{m}_i \left| \frac{1}{\hat{m}_i} \right|_{m_i} r_i \right|_M \quad (1)$$

where $\hat{m}_i = \frac{M}{m_i}$, $M = \prod_i m_i$ is the dynamic range, $\left| \frac{1}{\hat{m}_i} \right|_{m_i}$ is the multiplicative inverse of

\hat{m}_i , and $|\cdot|_m$ is the modulo operation. For the moduli $m_1 = 2^k + 1$, $m_2 = 2^k$,

$m_3 = 2^k - 1$, one has $\hat{m}_1 = 2^k(2^k - 1)$, $\hat{m}_2 = (2^{2k} + 1)$, $\hat{m}_3 = 2^k(2^k + 1)$. Andraos

and Ahmed show that the multiplicative inverses are given by $\left| \frac{1}{m_1} \right| = 2^{k-1} + 1$,

$\left| \frac{1}{m_2} \right| = 2^k - 1$ and $\left| \frac{1}{m_3} \right| = 2^{k-1}$, resulting in a simplified expression for the conversion

as given in eqn. (2).

$$X = \left[\frac{X}{2^k} \right] 2^k + r_2 \quad (2)$$

where:

$$\left\lceil \frac{X}{2^k} \right\rceil = |(C - r_1) + (B + A)|_{2^{2k-1}} \quad (3)$$

$$A = |(2^{2k-1} + 2^{k-1})r_3|_{2^{2k-1}} \quad (4)$$

$$B = |(2^{2k} - 2^k - 1)r_2|_{2^{2k-1}} \quad (5)$$

$$C = |(2^{2k-1} + 2^{k-1})r_1|_{2^{2k-1}} \quad (6)$$

Andraos and Ahmed [3] use only logic operations to evaluate A , B , and C . Since the evaluation of eqn. (2) is a simple concatenation of r_2 to $\left\lceil \frac{X}{2^k} \right\rceil$, the modulo $(2^{2k} - 1)$ summation of eqn. (3) is the only arithmetic operation required for the conversion. In [3] it is proposed to use four $2k$ bit binary adders, two of them in parallel, to evaluate this summation; this results in an efficient implementation.

Now we discuss improvements that can be made to this algorithm.

3.0 Improved Conversion Algorithm

First we show that the four numbers in the summation of eqn. (3) can be reduced to three numbers.

Let us consider the summation of the first two numbers

$$\begin{aligned} |C - r_1|_{2^{2k-1}} &= |(2^{2k-1} + 2^{k-1} - 1)r_1|_{2^{2k-1}} \\ &= |(2^{2k} - 2^{2k-1} + 2^{k-1} - 1)r_1|_{2^{2k-1}} \\ &= |(-2^{2k-1} + 2^{k-1})r_1|_{2^{2k-1}} \end{aligned} \quad (7)$$

From the fact that:

$$|2^{2k}|_{2^{2k-1}} = 1 \quad (8)$$

$|X \cdot 2^m|_{2^{2k-1}}$ can be simply implemented by an m -bit left cyclic shift.

Assuming that r_1 is expressed in $2k$ bits, where the $k-1$ most significant bits are zeros.

$$r_1 = \underbrace{0 \dots 0}_{k-1} c_k c_{k-1} \dots c_0. \quad (9)$$

and

$$\left| (-2^{2k-1} + 2^{k-1}) r_1 \right|_{2^{2k-1}} = \left| \overline{c_0} \underbrace{1 \dots 1}_{k-1} \overline{c_k \overline{c_{k-1}} \dots \overline{c_1}} + c_k c_{k-1} \dots c_0 \underbrace{0 \dots 0}_{k-1} \right|_{2^{2k-1}} \quad (10)$$

where the negative of a number modulo $(2^{2k} - 1)$ is the ones complement of that number.

We can rewrite the summation as shown in eqn. (11).

$$\left| C - r_1 \right|_{2^{2k-1}} = \left| \overline{c_0} c_{k-1} \dots c_0 \overline{\overline{c_{k-1}} \dots \overline{c_1}} + c_k \underbrace{1 \dots 1}_{k-1} \overline{c_k} \underbrace{0 \dots 0}_{k-1} \right|_{2^{2k-1}} \quad (11)$$

Assuming that the $2k$ bit expression of r_2 is given by:

$$r_2 = \underbrace{0 \dots 0}_k b_{k-1} b_{k-2} \dots b_0 \quad (12)$$

then:

$$\begin{aligned} B &= \left| (2^{2k} - 2^k - 1) r_2 \right|_{2^{2k-1}} \\ &= \left| (-2^k) r_2 \right|_{2^{2k-1}} \end{aligned} \quad (13)$$

or

$$\begin{aligned} B &= \overline{b_{k-1} b_{k-2} \dots b_0} 1 \dots 1 \\ &= \underbrace{0 \dots 0}_k \underbrace{1 \dots 1}_k \overline{b_{k-1} b_{k-2} \dots b_0} \underbrace{0 \dots 0}_k. \end{aligned} \quad (14)$$

Thus

$$|C - r_1 + B|_{2^{2k-1}} = \left| \begin{array}{l} \overline{c_0 c_{k-1} \dots c_0 \overline{c_{k-1} \dots c_1} + c_k \underbrace{1 \dots 1}_{k-1} \overline{c_k} \underbrace{0 \dots 0}_{k-1}} \\ + \underbrace{0 \dots 0}_k \underbrace{1 \dots 1}_k + \overline{b_{k-1} b_{k-2} \dots b_0} \underbrace{0 \dots 0}_k \end{array} \right|_{2^{2k-1}} \quad (15)$$

Let us first consider the summation of the second and third numbers on the right hand side. Except for the k -th least significant bit position, for every bit position of these two numbers, there is only one non-zero bit. The summation of the k -th least significant bit is the summation of 1 and $\overline{c_k}$; that yields a sum of c_k and a carry of $\overline{c_k}$ to the $k+1$ -th bit position. Since that bit of the second number is 1, the carry, $\overline{c_k}$, will propagate to the most significant bit position where, for the second number, this is c_k . The summation of $\overline{c_k}$ and c_k is 1 and no carry is produced. The result of the summation of the second and third numbers is therefore $\underbrace{1 c_k \dots c_k}_k \underbrace{1 \dots 1}_{k-1}$, and so the k bits starting from the k -th least significant bit to

the next most significant bit are all c_k . Since c_k and c_i , $i=0,1,\dots,k-1$, can never be 1 at the same time, they can be combined with the k bits in the same range of the first number on the right hand side of eqn. (15). Now we can write:

$$|C - r_1 + B|_{2^{2k-1}} = \left| \overline{c_0 c_{k-1} \dots \hat{c}_0 \overline{c_{k-1} \dots c_1} + 1 \underbrace{0 \dots 0}_k \underbrace{1 \dots 1}_{k-1}} \right. \\ \left. + \overline{b_{k-1} b_{k-2} \dots b_0} \underbrace{0 \dots 0}_k \right|_{2^{2k-1}} \quad (16)$$

where $\hat{c}_i = c_i \vee c_k$, $i=0,1,\dots,k-1$, (\vee denotes logic OR). Note that the k -th least significant bits of the second and the third numbers of the right hand side of eqn. (16) are zeros, and that, except for the most significant bit position, there is at least one zero bit in these two numbers for any bit position. Using the cyclic shifting property again, and proceeding as before, we can add these two number to obtain a single $2k$ bit number. Thus,

$$|C - r_1 + B|_{2^{2k-1}} = \left| \overline{c_0 c_{k-1} \dots \hat{c}_0 \overline{c_{k-1} \dots c_1}} \right. \\ \left. + \overline{b_{k-1} b_{k-2} \dots b_0 \overline{b_{k-1}}} \underbrace{b_{k-1} \dots b_{k-1}}_{k-1} \right|_{2^{2k-1}} \quad (17)$$

3.1 An example

To demonstrate the validity of the above reduction of three $2k$ bit numbers to two, we provide an example.

Let $k=3$, $2^{2k} - 1 = 63$, $r_1 = 3 = 0011$, and $r_2 = 3 = 011$. Then

$$\begin{aligned} |C - r_1 + B|_{2^{2k} - 1} &= |(2^5 + 2^2 - 1) \times 3 - 2^3 \times 3|_{63} \\ &= |105 - 24|_{63} \\ &= 18 \end{aligned}$$

Using the new algorithm, $\overline{c_0} \hat{c}_2 \hat{c}_1 \hat{c}_0 \overline{c_2} \overline{c_1} = 001110 = 14$, and

$b_2 \overline{b_1} \overline{b_0} \overline{b_2} b_2 b_2 = 000100 = 4$. Adding these two numbers together we obtain the correct result of 18.

4.0 Hardware implementation

To implement the modulo $(2^{2k} - 1)$ addition of three $2k$ -bit numbers efficiently, we may use $2k$ full adders as carry save adders (CSA) to convert the three $2k$ bit numbers into two. From eqn. (8), the carry-out from the most significant bit is fed to the least significant bit position. Then a fast $2k$ -bit carry propagate adder (CPA), with its carry-out connected to its carry-in, is used to perform the modulo $(2^{2k} - 1)$ addition of the two numbers to yield the final result. The structure is shown in Fig. 1. The same weighted bits from each number are the inputs to a full adder.

4.1 Removing redundancy

Using $2k$ bits to represent a number modulo $(2^{2k} - 1)$, there are two representations for zero; i.e. $0, 0, \dots, 0$ and $1, 1, \dots, 1$. This redundancy can be removed by eliminating the latter representation. The approach is shown in the following:

Let $a_{2k-1}a_{2k-2}\dots a_0$ be the final outputs from the $2k$ -bit CPA adder, and let:

$$a = a_{2k-1} \wedge a_{2k-2} \wedge \dots \wedge a_0$$

where \wedge represents logic AND. Then:

$$d_i = \bar{a} \wedge a_i = \begin{cases} 0 & \text{if } a_i = 1; \quad i = 0, 1, \dots, 2k-1 \\ a_i & \text{otherwise} \end{cases}$$

and the number, $d_{2k-1}d_{2k-2}\dots d_0$ will give the final result where the redundant representation of zero has been eliminated.

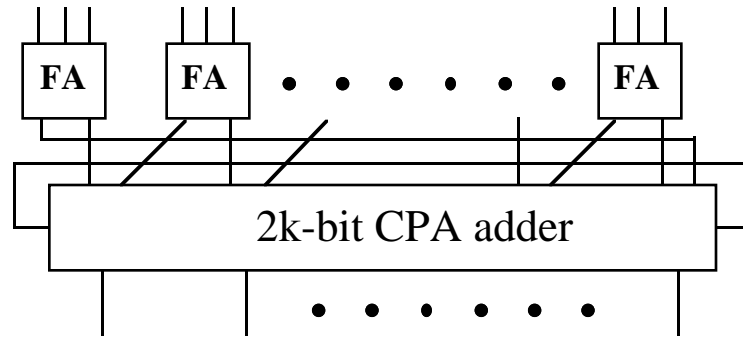


Figure 1. The implementation of modulo $(2^{2k} - 1)$ addition of three $2k$ -bit numbers

4.2 Performance evaluation and comparison

In order to evaluate the cost of the converter, and to compare our technique with other approaches, we need to establish the CPA adder technology to be used. We have performed the following analysis assuming the availability of a fast carry look-ahead adder, such as discussed in reference [8].

For a k -bit CPA adder, we evaluate the delay, D_k , and the silicon area, A_k , by:

$$D_k = (\log_2 k + 1)D_{FA} \tag{18}$$

and

$$A_k = (k \log_2 k)A_{FA} \tag{19}$$

where D_{FA} and A_{FA} are the delay and the silicon area for a full adder, respectively.

The elimination of the redundant representation of zero is a common issue for all approaches we are going to compare. Therefore we have not included this step in the following estimation.

The delay of a modulo $(2^{2k} - 1)$ adder includes the delay of a $2k$ -bit CPA adder, D_{2k} , together with the delay of $2k$ -bit carry propagation, which we assume to be proportional to D_{2k} . Thus, the delay of a modulo $(2^{2k} - 1)$ adder is given by:

$$\begin{aligned} D_{M2k} &= (1 + \beta)D_{2k} \\ &= (1 + \beta)(\log_2 k + 2)D_{FA} \end{aligned} \quad (20)$$

where the constant, β , is less than 1. The silicon area for a modulo $(2^{2k} - 1)$ adder, as implemented by our technique, is identical to that for a $2k$ -bit CPA adder.

With the above analysis, we may readily obtain the delay and silicon area for the proposed converter as:

$$\begin{aligned} D &= D_{M2k} + D_{FA} \\ &= [(1 + \beta)(\log_2 k + 2) + 1]D_{FA} \end{aligned} \quad (21)$$

and

$$\begin{aligned} A &= A_{2k} + 2kA_{FA} \\ &= 2k(\log_2 k + 2)A_{FA} \end{aligned} \quad (22)$$

In the following, we evaluate the delay and silicon area for the converters given in [3] and [7] for comparison.

The converter described in [3] requires three $2k$ -bit CPA adders, two of which operate in parallel, one modulo $(2^{2k} - 1)$ adder, and $2k-1$ AND gates, which are cascaded in $\log_2(2k)$ stages to provide the input to the modulo $(2^{2k} - 1)$ adder. The delay is estimated by eqn. (23):

$$\begin{aligned}
 D' &= 2D_{2k} + D_{M2k} + \log_2(2k)D_{AND} \\
 &= [(3 + \beta)\log_2 k + 6 + \beta]D_{FA} + (\log_2 k + 1)D_{AND}
 \end{aligned} \tag{23}$$

where D_{AND} is the delay of an AND gate.

We assume that the modulo $(2^{2k} - 1)$ adder in [3] is implemented with our technique, as shown in Figure 1. Then the silicon area for the converter in [3] can be estimated by eqn. (24):

$$\begin{aligned}
 A' &= 4A_{2k} + (2k - 1)A_{AND} \\
 &= 8k(\log_2 k + 1)A_{FA} + (2k - 1)A_{AND}
 \end{aligned} \tag{24}$$

where A_{AND} is the area of an AND gate.

The converter in [7] requires two modulo $(2^k - 1)$ subtractors, one modulo $(2^k + 1)$ subtractor, and one modulo $(2^{2k} - 1)$ adder. To implement a modulo $(2^k - 1)$ or $(2^k + 1)$ subtractor requires, as suggested in [7], a k -bit adder followed by a k -bit carry corrector, as shown in Fig. 1 of [7]. This only requires the same hardware and delay as a k -bit adder. To improve the speed we assume that fast CPA adders are employed. Thus the delay and silicon area for the converter in [7] can be estimated by eqn. (25) and eqn. (26):

$$\begin{aligned}
 D'' &= 4D_k + (1 + \beta)D_{2k} \\
 &= [(5 + \beta)\log_2 k + 6 + 2\beta]D_{FA}
 \end{aligned} \tag{25}$$

$$\begin{aligned}
 A'' &= 4A_k + A_{2k} \\
 &= 2k(3\log_2 k + 1)A_{FA}
 \end{aligned} \tag{26}$$

Table 1 provides a comparison between the new converter and the designs in references [3] and [7] for $k \in \{4, 8, 16\}$, where the delay and area are normalized to the delay and area of a full adder, respectively. The delay and the area for the AND gates in eqn. (23) and eqn. (24) are ignored and we assume that $\beta = 0.7$. The Area Delay products in Table 1, give an overall performance comparison between the three converters. We can clearly observe a hardware saving of over 40% while almost halving the delay.

Table 1. Comparative performance with the converters in [3] and [7]

k	D	D'	D''	A	A'	A''	AD	$A'D'$	$A''D''$
4	7.8	14	18.8	33	96	56	257	1354	1053
8	9.5	17.8	24.5	81	256	160	770	4557	3920
16	11.2	21.5	30.2	193	640	416	2162	13760	12563

5.0 Conclusions

An efficient residue to binary converter for the modulus set $(2^n + 1, 2^n, 2^n - 1)$ is proposed. The efficiency is obtained by an improvement on an existing algorithm, and the introduction of a more efficient hardware implementation. Comparing to two converters, which are recognized as among the more efficient of the previously published art, our new converter architecture provides at least a 40% hardware saving and a 45% reduction in the delay.

6.0 Acknowledgments

The authors acknowledge support from the Natural Science and Engineering Research Council of Canada, and the Micronet Network of Centres of Excellence for funding this work. Design tools have been provided by the Canadian Microelectronics Corporation.

7.0 References

- [1] F. J. Taylor and A. S. Ramnayan, "An efficient residual to decimal converter, *IEEE Trans. Circuits & Systems*, vol. CAS-28, pp. 1164-1169, 1981
- [2] B. Barnardson, "Fast memoryless, over 64 bits, residual to binary converter," *IEEE Trans. Circuits & Systems*, vol. CAS-32, pp. 298-300, 1985.
- [3] S. Andraos and H. Ahmed, "A new efficient memoryless residual to binary converter", *IEEE Trans. Circuits & Systems*, vol. CAS-35, pp. 1441-1444, 1988.

- [4] K. M. Ibrahim and S, N. Saloumm, "An efficient residue to binary converter design," *IEEE Trans. Circuits & Systems*, vol. CAS-35, pp. 1156-1158, 1988.
- [5] G. Bi and E. V. Jones, "Fast conversion between binary and residual numbers", *Electronics Letters*, vol. 24, pp. 1195-1197, 1988.
- [6] A. Dhurkadas, "Comments on 'An efficient residue to binary converter design'," *IEEE Trans. Circuits & Systems*, vol. CAS-37, pp. 849-850, 1990.
- [7] B. Vinnakota and V. V. B. Rao, "Fast conversion techniques for binary-residue number systems" *IEEE Trans. Circuits & Systems*, vol. CAS-41, pp. 927-929, 1994.
- [8] T. Lynch and E. E. Swartzlander, "A Spanning Tree Carry Lookahead Adder" *IEEE Trans. Comput.* vol. C-41, pp. 931-939, 1992.