

An Efficient 3-Modulus Residue to Binary Converter

Zhongde Wang*, G.A. Jullien and W.C. Miller
VLSI Research Group
University of Windsor, Windsor, Ontario, Canada N9B 3P4

*. Now with Genesis Microchip, Markham, On. L3R 8G5, Canada

Abstract

An efficient algorithm, which converts the residue numbers, with moduli set $\{2^N + 1, 2^N, 2^N - 1\}$ into an equivalent binary number, is presented; the algorithm improves on a previously published technique. A novel hardware implementation is also discussed. In comparison to the cited converter, the hardware savings are in excess of 66% and a reduction of 30% in the critical path is obtained.

1. Introduction

The three modulus, $(2^N + 1, 2^N, 2^N - 1)$, 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 compute intensive task. Among the 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 in [3], as a result, hardware savings in excess of 66% and a reduction of 30% in the critical path is obtained.

2. Andraos and Ahmed's Algorithm

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

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

where $\hat{m}_i = \frac{M}{m_i}$. $M = \prod_i m_i$ is the dynamic range. $\left| \frac{1}{\hat{m}_i} r_i \right|_{m_i}$ is

the multiplicative inverse of \hat{m}_i , and $|\cdot|_m$ is the modulo operation.

For the moduli set $m_1 = 2^k + 1$, $m_2 = 2^k$,

$m_3 = 2^k - 1$, 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 [3] 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 Eq. (2).

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

where

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

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

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

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

Andraos and Ahmed [3] use purely logic operations to evaluate A , B , and C . Since the evaluation of Eq. (2) is a simple

concatenation of r_2 to $\left[\frac{X}{2^k} \right]$, the modulo $(2^{2k} - 1)$ summa-

tion of Eq. (3) is the only arithmetic operation required for the conversion. In [3] it is proposed to use four $2k$ bit binary adder, 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. Improved Conversion Algorithm

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

Let us consider the summation of the first two numbers

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

From the fact that:

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

$\lfloor X \cdot 2^m \rfloor_{2^{2k-1}}$ 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

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

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).

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

Assuming 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

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

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 $1 c_k c_k \dots c_k \underbrace{1 \dots 1}_{k-1}$. 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 Eq. (15). Now we can write:

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

where $\hat{c}_i = c_i \vee c_k$, $i=0,1,\dots,k-1$, (\vee denotes logic OR).

Notice 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,

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

4. An Example

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

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

Then

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

Using the new algorithm, $\overline{c_0}\hat{c}_2\hat{c}_1\hat{c}_0\overline{c_2c_1}$ = 001110 = 14. and $b_2\overline{b_1}\overline{b_0}\overline{b_2}b_2b_2$ = 000100 = 4. Adding these two numbers together we obtain the correct result of 18.

5. Hardware Implementation

To implement the modulo $(2^{2k} - 1)$ addition of three $2k$ -bit numbers efficiently, we may use $2k$ full adders to convert the three $2k$ bit numbers into two. From Eq. (8), the carry-out from the most significant bit is fed to the least significant bit position. Then a fast $2k$ -bit binary adder, 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.

6. Removing Redundancy

Using $2k$ bits to represent a number modulo $(2^{2k} - 1)$, there are two redundant representations for zero. We require to remove the redundancy by eliminating the representation $\underbrace{11\dots 1}$; this can be realized as follows.

Let $a_{2k-1}a_{2k-2}\dots a_0$ be the final outputs from the $2k$ -bit adder, and a be the logic AND of all $2k$ output bits from the adder. Then the final result is given by $d_{2k-1}d_{2k-2}\dots d_0$ where $d_i = \overline{a} \wedge a_i$, and \wedge represents logic AND.

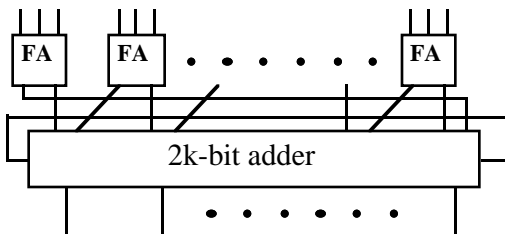


Figure 1. The implementation of modulo $(2^{2k} - 1)$ addition of three $2k$ -bit numbers

7. Performance Evaluation

In order to evaluate the cost of the converter, we need to establish the 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 $2k$ -bit adder, we evaluate the delay, D_{2k} , and the silicon area, A_{2k} , by:

$$D_{2k} = 2\log_2(2k) \times D_{FA} \quad (19)$$

and

$$A_{2k} = 2k\log_2(2k) \times A_{FA} \quad (20)$$

where D_{FA} and A_{FA} are the delay and the silicon area for a full adder, respectively; the delay and area of our new converter are given by:

$$D = (2k\log_2(2k) + 1)D_{FA} + D_P \quad (21)$$

and

$$A = 2k(\log_2(2k) + 1)A_{FA} \quad (22)$$

respectively, where D_P is the delay of a $2k$ bit carry propagation.

For comparison, the delay and the area of the converter given in [3] are given by

$$D_1 = 6\log_2(2k)D_{FA} \quad (23)$$

and

$$A_1 = 8k\log_2(2k)A_{FA} \quad (24)$$

respectively. The delay of a $2k$ -bit carry propagation is certainly shorter than the delay of a $2k$ -bit adder. One may estimate that the ratio D/D_1 will be in the range from 0.6 to 0.7.

On the other hand, the ratio A/A_1 is less than 1/3 for $k > 4$.

8. Conclusions

In conclusion, an efficient residue to binary converter for the modulus set $(2^N + 1, 2^N - 1)$ is proposed. The efficiency is obtained by an improvement on an existing algorithm, and the introduction of a more efficient hardware implementation. Comparing to one of the most efficient of the published converters, more than 66% of the hardware is saved, and the critical delay path is reduced by over 30%.

9. 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.

10. 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, "Comment 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.