Modified Booth encoding modulo $(2^n - 1)$ multipliers

Lei Li^{1a)}, Jianhao Hu², and Yiou Chen²

¹ Research Institute of Electronic Science and Technology, University of Electronic Science and Technology of China

No.2006, Xiyuan Avenue, High-Tech West Zone, Chengdu, 611731, Sichuan, China

² National Key Lab. of Science and Technology on Communication, University of Electronic Science and Technology of China

No.2006, Xiyuan Avenue, High-Tech West Zone, Chengdu, 611731, Sichuan, China

a) leilee@uestc.edu.cn

Abstract: $(2^n - 1)$ is one of the most commonly used moduli in Residue Number Systems. In this express, we propose a novel Booth encoding architecture. Based on the proposed Booth encoding architecture, we can design high speed and high-efficient modulo $(2^n - 1)$ multipliers, which are the fastest among all known modulo $(2^n - 1)$ multipliers. The performance and the efficiency of the proposed multipliers are evaluated and compared with the earlier fastest modulo $(2^n - 1)$ multipliers, based on a simple gate-count and gate-delay model. These results reveal that the proposed multipliers lead to average approximately 14% faster than the fastest known modulo $(2^n - 1)$ multipliers.

Keywords: Residue Number Systems (RNS), multiplier

Classification: Integrated circuits

References

- [1] M. Soderstrand, W. Jenkins, G. Jullien, and G. Taylor, “Residue number system arithmetic: modern applications in digital signal processing,” IEEE Press, Piscataway, NJ, USA, 1986.
- [2] L. S. Didier and P. Y. Rivaille, “A generalization of a fast rns conversion for a new 4-modulus base,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 56, no. 1, pp. 46–50, Jan. 2009.
- [3] C. Efstathiou, H. T. Vergos, and D. Nikolos, “Modified Booth Modulo $2^n - 1$ Multipliers,” *IEEE Trans. Comput.*, vol. 53, no. 3, pp. 370–374, March 2004.
- [4] R. Zimmerman, “Efficient VLSI Implementation of Modulo $(2^n \pm 1)$ Addition and Multiplication,” *Proc. 14th IEEE Symp. Computer Arithmetic*, pp. 158–167, April 1999.
- [5] K. Manochchri, S. Pourmozafari, and B. Sadeghian, “Efficient Methods in Converting to Modulo $2^n + 1$ and $2^n - 1$,” *Proc. IEEE Information Technology: New Generations*, pp. 178–185, 2006.

1 Introduction

A Residue Number System (RNS) is defined by a pair-wise prime moduli set $\{m_1, m_2, \dots, m_k\}$ [1]. An integer X is represented in RNS as $\{x_1, x_2, \dots, x_k\}$, where x_i is the nonnegative remainder of X division m_i : $x_i = X \bmod m_i \equiv \langle X \rangle_{m_i}$. The dynamic range of the RNS is $M_K = x_1 \cdot x_2 \cdots x_k$. The integers in the range of $[0, M_K)$ can be uniquely represented in this RNS. Moreover, the operations addition, subtraction and multiplication in binary number systems become modulus addition, modulus subtraction and modulus multiplication in RNS, respectively. The operations in binary number systems and the modulus operations in RNS have the following relations as shown in (1) (“ Θ ” represents addition, subtraction and multiplication):

$$\begin{cases} X \xrightarrow{RNS} \{x_1, x_2, \dots, x_k\}; \\ Z \xrightarrow{RNS} \{z_1, z_2, \dots, z_k\}; \end{cases} \quad \text{define: } Y = X \Theta Z; \quad (1)$$

$$Y \xrightarrow{RNS} \{y_1, y_2, \dots, y_k\} = \{\langle x_1 \Theta z_1 \rangle_{m_1}, \langle x_2 \Theta z_2 \rangle_{m_2}, \dots, \langle x_k \Theta z_k \rangle_{m_k}\}$$

That is, in a RNS, the operations between residues are performed in parallel units, each one handling small residues.

Moduli of the forms $\{2^n, 2^n - 1, 2^n + 1\}$ and $\{2^n, 2^n - 1, 2^{n-1} - 1\}$ are the most commonly used moduli in RNS, because they have lower implementation difficulty and computational complexity than other moduli forms, when considering the *area* $\times$ *time*² product, and also they can offer efficient converters from and to the binary system [2]. Therefore, modulo $(2^n - 1)$ has been widely used in many applications and it is a very interesting issue to find the new method to design efficient modulo $(2^n - 1)$ multipliers.

In recent years, several architectures have been proposed to design fast modulo $(2^n - 1)$ multipliers [3–5]. The most efficient modulo $(2^n - 1)$ multipliers were introduced in [3]. To our knowledge, the modulo $(2^n - 1)$ multipliers in [3] are the fastest known modulo $(2^n - 1)$ multipliers.

In this express, we propose a novel Booth encoding, which is much faster than ever before. Static analysis demonstrates that the proposed multipliers offer significant savings in execution delay against that in [3–5] and achieve high *area* $\times$ *time*² efficiency.

2 The proposed modified Booth modulo $(2^n - 1)$ multiplier

In this section, we present the proposed architecture for our modified Booth modulo $(2^n - 1)$ multipliers. Suppose that A is the multiplicand and B the multiplier and we have $A = a[n-1] \cdots a[i] \cdots a[0]$, and $B = b[n-1] \cdots b[i] \cdots b[0]$ with n -bit numbers in modulo $(2^n - 1)$ representation. And the multiplier B can be rewritten as:

$$B = \sum_{i=0}^{n-1} b[i] \times 2^i$$

$$= \begin{cases} b[n] \times 2^n + \sum_{i=0}^{n-1} b[i] \times 2^i |_{b_n=0, n \text{ is even}} \\ b[n+1] \times 2^{n+1} + b[n] \times 2^n + \sum_{i=0}^{n-1} b[i] \times 2^i |_{b_{n+1}=0, b_n=0, n \text{ is odd}} \end{cases} \quad (2)$$

Setting $E[k] = b[k-1] + b[k] - 2 \times b[k+1]$, we can get the formation of partial products showed in Table I from [3], where

$$PP[k] = pp[k][n-1] \cdots pp[k][i] \cdots pp[k][0] = \langle E[k] \times A \times 2^{2k} \rangle_{2^{n-1}} \quad (3)$$

By performing logic addition and logic simplification, the terms of the sixth row with $a[i]$ or $\overline{a[i]}$ (the notation $\overline{x}$ is used to denote the complement of x) in Table I can be formulated as:

$$\begin{aligned} \psi[k][i] &= \left(\overline{b[2k+1]} \overline{b[2k]} b[2k-1] + \overline{b[2k+1]} b[2k] \overline{b[2k-1]} \right) a[i] \\ &\quad + \left(b[2k+1] \overline{b[2k]} b[2k-1] + b[2k+1] b[2k] \overline{b[2k-1]} \right) \overline{a[i]} \\ &= (b[2k] \oplus b[2k-1]) (b[2k+1] \oplus a[i]) \end{aligned} \quad (4)$$

and the terms of the sixth row with $a[i-1]$ or $\overline{a[i-1]}$ in Table I can be expressed as:

$$\begin{aligned} \xi[k][i-1] &= \overline{b[2k+1]} b[2k] b[2k-1] a[i-1] + b[2k+1] \overline{b[2k]} \overline{b[2k-1]} \overline{a[i-1]} \\ &= (b[2k] \odot b[2k-1]) \left(b[2k+1] \overline{b[2k]} \overline{a[i-1]} + \overline{b[2k+1]} b[2k] a[i-1] \right) \end{aligned} \quad (5)$$

Table I. Formation of partial products

$b[2k+1]$	$b[2k]$	$b[2k-1]$	$E[k]$	$PP[k]$	$pp[k][i]$
0	0	0	0	0	0
0	0	1	+1	$\langle 2^{2k} A \rangle_{2^{n-1}}$	$a[i]$
0	1	0	+1	$\langle 2^{2k} A \rangle_{2^{n-1}}$	$a[i]$
0	1	1	+2	$\langle 2^{2k+1} A \rangle_{2^{n-1}}$	$a[i-1]$
1	0	0	-2	$\langle -2^{2k+1} A \rangle_{2^{n-1}}$	$\overline{a[i-1]}$
1	0	1	-1	$\langle -2^{2k} A \rangle_{2^{n-1}}$	$\overline{a[i]}$
1	1	0	-1	$\langle -2^{2k} A \rangle_{2^{n-1}}$	$\overline{a[i]}$
1	1	1	0	+0	1

The term $b[2k+1]b[2k]b[2k-1]$ is needed to added to $\psi[k][i]$ or $\xi[k][i-1]$ to produce $pp[k][i]$. The term $b[2k+1]b[2k]b[2k-1]$ is very difficult to merge with $\psi[k][i]$ because of $\psi[k][i]$ with the term $b[2k] \oplus b[2k-1]$, while it is very easy to merge with $\xi[k][i-1]$ because of $\xi[k][i-1]$ with the item $b[2k] \odot b[2k-1]$. Therefore, we have

$$\begin{aligned} \zeta[k][i-1] &= \xi[k][i-1] + b[2k+1]b[2k]b[2k-1] \\ &= (b[2k] \odot b[2k-1]) \left(b[2k+1] \overline{a[i-1]} + b[2k] a[i-1] \right) \end{aligned} \quad (6)$$

Fig. 1. Booth encoder

Fig. 2. Booth selector

Letting $h[k] = b[2k] \oplus b[2k-1]$, $l[k][i] = b[2k+1] \oplus a[i]$ and $r[k][i-1] = b[2k+1]\overline{a[i-1]} + b[2k]a[i-1]$, we get

$$pp[k][i] = \psi[k][i] + \zeta[k][i-1] = h[k] \cdot l[k][i] + \overline{h[k]} \cdot r[k][i-1] \quad (7)$$

We consider $h[k] = b[2k] \oplus b[2k-1]$ is Booth encoder (BE) logic equation. Fig. 1 plots the implementation adopted for the Booth encoder. Obviously, the expression for the term $pp[k][i]$ can be implemented with a XNOR and two 2 : 1 inverting multiplexers. Fig. 2 presents the implementation adopted for the Booth selector (BS), where MXI is a 2 : 1 inverting multiplexer.

The control signals produced by the multiplier B , along with the multiplicand A , are used to form partial products and each Booth code produces a partial product. Every bit of each partial product is produced by a Booth selector block, just as shown in Fig. 2. The produced partial products are reduced to two operands with CSA arrays or adder trees (Wallace trees), sometimes in several stages. “The carry output at the most significant bit position of each stage has a weight of 2^n , which, in modulo $(2^n - 1)$ arithmetic, is equal to 1. Therefore, these carries are added in an end-around carry way to the least significant bit position of the operands of the next stage.” [3] These two resulting operands are added in a parallel modulo $(2^n - 1)$ adder to compute the final result.

3 Analysis and Comparison

In this section, we demonstrate the improved performance of the proposed multipliers against the multiplier designs of [3-5].

The proposed design requires an area equal to

$$A_{Pro} = \left\lceil \frac{n}{2} \right\rceil A_{BE,Pro} + n \left\lceil \frac{n}{2} \right\rceil A_{BS,Pro} + n \left(\left\lceil \frac{n}{2} \right\rceil - 2 \right) A_{FA} + A_{PA_n} \quad (8)$$

where $A_{BE,Pro}$ is the area of a BE, $A_{BS,Pro}$ is the area of a BS, A_{FA} is the area of a full adder (FA), A_{PA_n} is the area of a modulo $(2^n - 1)$ adder

(PA(n)), $\lceil * \rceil$ is the smallest integer larger than or equal to *. The delay of the proposed multiplier equals to

$$T_{Pro,CSA} = T_{BS,Pro} + \left(\left\lceil \frac{n}{2} \right\rceil - 2 \right) T_{FA} + T_{PA_n} \quad (9)$$

when a CSA array is used and equals to

$$T_{Pro,WAL} = T_{BS,Pro} + k \left(\left\lceil \frac{n}{2} \right\rceil \right) T_{FA} + T_{PA_n} \quad (10)$$

when Wallace trees are used, where $T_{BS,Pro}$ is the delay of a BS, T_{FA} is the delay of a FA, T_{PA_n} is the delay of a modulo $(2^n - 1)$ adder (PA(n)) and $k(x)$ equals to 0, 1, 2, 3, 4, 4, 6 and 6 when x is 2, 3, 4, 5, 8, 9, 16 and 17, respectively [3].

Considering the $area \times time^2$ product, the ratio of efficiency of the proposed multiplier to the reference multiplier can be defined as:

$$E_{Ref,Pro} = AE_{Ref,Pro} \times TE_{Ref,Pro}^2 = \frac{A_{Ref}}{A_{Pro}} \times \left(\frac{T_{Ref}}{T_{Pro}} \right)^2 \quad (11)$$

The area of the reference multiplier [3] can be given as

$$A_{[3]} = \left\lceil \frac{n}{2} \right\rceil A_{BE} + n \left\lceil \frac{n}{2} \right\rceil A_{BS} + n \left(\left\lceil \frac{n}{2} \right\rceil - 2 \right) A_{FA} + A_{PA_n} \quad (12)$$

The delay of the reference multiplier [3] equals to

$$T_{[3],CSA} = T_{BE} + T_{BS} + \left(\left\lceil \frac{n}{2} \right\rceil - 2 \right) T_{FA} + T_{PA_n} \quad (13)$$

when a CSA array is used and

$$T_{[3],WAL} = T_{BE} + T_{BS} + k \left(\left\lceil \frac{n}{2} \right\rceil \right) T_{FA} + T_{PA_n} \quad (14)$$

when Wallace trees are used.

The area of the reference multiplier [4] can be given as

$$A_{[4]} = \left(\left\lfloor \frac{n}{2} \right\rfloor + 1 \right) A_{BE,[3]} + n \left(\left\lfloor \frac{n}{2} \right\rfloor + 1 \right) A_{BS,[3]} + n \left(\left\lfloor \frac{n}{2} \right\rfloor - 1 \right) A_{FA} + A_{PA_n} \quad (15)$$

where $\lfloor * \rfloor$ is the largest integer less than or equal to *.

The delay of the reference multiplier [4] equals to

$$T_{[4],CSA} = T_{BE} + T_{BS} + \left(\left\lfloor \frac{n}{2} \right\rfloor - 1 \right) T_{FA} + T_{PA_n} \quad (16)$$

when a CSA array is used and

$$T_{[4],WAL} = T_{BE} + T_{BS} + k \left(\left\lfloor \frac{n}{2} \right\rfloor + 1 \right) T_{FA} + T_{PA_n} \quad (17)$$

when Wallace trees are used.

The area of the reference multiplier [5] can be given as

$$A_{[5]} = n^2 + n(n-2)A_{FA} + \frac{3}{2}A_{PA_n} \quad (18)$$

The delay of the reference multiplier [5] equals to

$$T_{[5],CSA} = 1 + (n-2)T_{FA} + \frac{3}{2}T_{PA_n} \quad (19)$$

when a CSA array is used and

$$T_{[5],WAL} = 1 + k(n+1)T_{FA} + \frac{3}{2}T_{PA_n} \quad (20)$$

when Wallace trees are used.

Based on the simple gate-count and gate-delay model used in [3], the following approximations can be obtained: $A_{BE,Pro}$ equals to 2 equivalent gates, $A_{BS,Pro}$ equals to 6 equivalent gates, $T_{BS,Pro}$ equals to 4 time units, A_{BE} equals to 5 equivalent gates, T_{BE} equals to 3 time units, A_{BS} equals to 5 equivalent gates, T_{BS} equals to 4 time units, A_{FA} equals to 7 equivalent gates, T_{FA} equals to 4 time units, A_{PA_n} equals to $3n \log_2 n + 4n$ equivalent gates, T_{PA_n} equals to $2 \log_2 n + 3$ time units. Based on these approximations, the delay and area comparisons of the proposed designs against those of [3–5] for $n = 4, 8$, and 16 are studied, with either a CSA array or a Wallace tree partial products reduction scheme used. Static analysis demonstrates that the proposed multipliers can achieve up to 21.4%, 38.9% and 53.2% faster designs than those of [3] [4] [5], respectively. On the average, the proposed multipliers are faster than those of [3] [4] [5] by 13.7%, 26.4% and 45.3%, with a CSA used and by 14.2%, 24.2% and 36.6% with Wallace trees used, respectively. Compared against [3], the proposed multipliers can achieve the faster speed because the proposed Booth encoding architecture can shorten the critical paths in the Booth encoding. The Booth encoding architecture in [3] has a delay of 7 time units, while the proposed Booth encoding architecture just has a delay of 4 time units. Compared against [4], the proposed multipliers can achieve the faster speed because the proposed Booth encoding architecture can shorten the critical paths in the Booth encoding and the proposed multipliers have less FA stages than those of [4]. Compared to those of [3], the proposed multipliers have a more 4.6% area overhead on the average. The proposed multipliers are 19.1% and 26.0% more area efficient than those of [4][5] on the average. There is a very significant improved performance of the proposed multipliers against those of [5]. The proposed multipliers employ Booth encoding to reduce the partial products, while the multipliers in [5] do not employ Booth encoding and have one extra modulo $(2^n - 1)$ adder using half adder. The proposed multipliers have $\lceil n/2 \rceil$ partial products, while the multipliers in [5] have n partial products. The ratios of efficiency of the proposed multipliers to the multipliers in [3][4][5] are 1.13–1.59, 1.26–4.15 and 2.31–6.55, respectively.

4 Conclusion

In this express, we have presented a new architecture for modified Booth modulo $(2^n - 1)$ multipliers. The proposed multipliers compare favorably, with respect to speed and efficiency, against the known modulo $(2^n - 1)$ multipliers. The proposed multipliers can lead to average to approximately 14% faster than the fastest known modulo $(2^n - 1)$ multipliers in [3]. The ratios of efficiency of the proposed multipliers to the multipliers in [3][4][5] are 1.13–1.59, 1.26–4.15 and 2.31–6.55, respectively.

Acknowledgments

The authors would like to thank ASIC Team for providing advice and discussion. This work was supported by National Natural Science Foundation of China (NSFC) under Grant 60873076 and 61076096.