INTEGRATION, the VLSI journal 47 (2014) 140–147

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

Efficient modulo 2^n+1 multiply and multiply-add units based on modified Booth encoding

Constantinos Efstathiou [a], N. Moshopoulos [b,*], N. Axelos [b,1], K. Pekmestzi [b]

[a] Department of Informatics, Technological Institute of Athens, 12210 Athens, Greece
[b] Department of Electrical and Computer Engineering, National Technical University of Athens, 15780 Athens, Greece

ARTICLE INFO

Article history:
Received 2 August 2012
Received in revised form
7 April 2013
Accepted 7 April 2013
Available online 22 April 2013

Keywords:
Modulo
Multipliers
Fused multiply-add units
Residue number system
RNS
Modified Booth multiplication

ABSTRACT

In this work a new efficient modulo 2^n+1 modified Booth multiplication algorithm for both operands in the weighted representation is proposed. Furthermore, the same algorithm is extended to realize modulo 2^n+1 multiply-add units. The derived partial products are reduced by an inverted end around carry-save adder tree to two operands, which are finally added by a modulo 2^n+1 adder. The performance and efficiency of the proposed multipliers are evaluated and compared against the earlier modulo 2^n+1 multipliers, based on a single gate level model. Comparisons based on experimental CMOS implementations for both the multiply and multiply-add units are also given. The proposed multipliers yield area and power savings by an average of 15% and 10% respectively, while the corresponding area and power savings of the proposed multiply-add units are 14% and 21% respectively.

1. Introduction

The Residue Number System (RNS) [1] reduces the delay of carry propagation, thus offering significant speedup over the conventional binary system. This characteristic is advantageous when repetitive arithmetic operations on long operands have to be performed. RNS has been adopted in the design of Digital Signal Processors (DSP) [2]. The low power consumption of RNS compared to conventional arithmetic circuits for the implementation of Finite Impulse Response (FIR) filters is presented in [3]. Discrete Cosine Transform (DCT) processors [4], communication components [5], cryptography [6,7] and other DSP applications [8] utilize efficiently the RNS. RNS can also be used in the design of arithmetic circuits that are variation tolerant [9]. Therefore, RNS may be an interesting candidate for building processing circuits in deep submicron technologies.

The moduli set $\langle 2^n-1,\ 2^n,\ 2^n+1\rangle$ and its extensions have received significant attention because they offer simple and efficient implementations [10].

In many RNS based systems modulo 2^n+1 units become a bottleneck, as they have to deal with $(n+1)$ bit wide operands, whereas the remaining units handle operands with length lower or equal to n bits.

The diminished-1 representation of binary numbers was introduced in [11] to speed up modulo 2^n+1 arithmetic operation. Since only n bits are required for the representation of the magnitude of any number $A\in(0,\ 2^n]$, the diminished-1 representation can lead to implementations with delay and area comparable to that of the modulo 2^n-1, 2^n units. Efficient diminished-1 modulo 2^n+1 arithmetic units have been proposed in [12–21]. Among them, the diminished-1 modulo 2^n+1 multipliers in [16,18] use Modified Booth (MB) encoding, while the multipliers proposed in [17] are the most efficient with conventional (without MB encoding) tree architectures. In [19], a modulo 2^n+1 MB multiplier is proposed, with one factor in the diminished-1 representation, and the second in the weighted representation. Diminished-1 modulo 2^n+1 fused multiply-add units, are proposed in [20,21].

The need for area, time and power consuming translators from the weighted to the diminished-1 representation and vice versa makes the design of weighted modulo 2^n+1 functional units a preferable candidate for many applications. Weighted modulo 2^n+1 multipliers have been proposed in [16,22–24]. The weighted modulo 2^n+1 multipliers, proposed in [16,23] use MB encoding, while those in [22] have a conventional tree architecture. Among the multipliers using MB encoding those in [23] are the most efficient.

Efficient fused multiply-add units which perform the operation $A\times B+D$ in one cycle are included in modern microprocessors and digital signal processors [25]. Many DSP algorithms have been

* Corresponding author. Tel.: +30 210 4955 093.
E-mail addresses: nikos@microlab.ntua.gr, nmoshop@gmail.com (N. Moshopoulos).

[1] The author contributed to this work while at the Department of Electrical and Computer Engineering, National Technical University of Athens (January—August 2011).

0167-9260/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.vlsi.2013.04.001

rewritten to take advantage of the presence of these units. Weighted modulo 2^n+1 fused multiply-add units, based on conventional tree architecture are proposed in [24]. Fused multiply-add units are also known in the literature as multiply-accumulate (MAC) units.

In this work we propose a new modulo 2^n+1 modified Booth multiplication algorithm for both operands in the weighted representation. Fused modulo 2^n+1 multiply-add units based on this algorithm are also described. The proposed multipliers have similar architecture with those in [23], however the new multipliers are based on a different algorithm that yield area and power savings. Additionally the proposed design is extended to include the efficient implementation of the fused multiply-add operation. The proposed multipliers and multiply-add units compared respectively against the multipliers and the multiply-add units based on conventional architectures [22,24] require less area and consume less power, while operating at the same speed.

The rest of the paper is organized as follows: in Section 2, the design of the modulo 2^n+1 multipliers and multiply-add units is presented. Section 3 includes implementation details. In Section 4, we estimate the complexity of the proposed multipliers and multiply-add units and compare them against earlier published designs. In Section 5 our conclusions are drawn.

2. Proposed designs

2.1. Design of modulo 2^n+1 multipliers

Let $A = a_n a_{n-1} \cdots a_1 a_0$, $B = b_n b_{n-1} \cdots b_1 b_0$ be the weighted representations of two numbers in the range $[0,\ 2^n+1)$, and $Q = |A \times B|_{2^n+1}$ their product modulo 2^n+1. In the following text the notation $X_{a:b}$ represent bits $x_a x_{a-1} \cdots x_b$ of an operand X, $(a > b)$. For the product Q of A, B we have that

$$Q = |(A+1) \times B - B|_{2^n+1} = |(a_n 2^n + A_{n-1:0} + 1) \times B - B|_{2^n+1} \tag{1}$$

Since $|2^n|_{2^n+1} = |-1|_{2^n+1}$, the following relation holds:

$$Q = |(A_{n-1:0} + 1) \times |B|_{2^n+1} - a_n B - B|_{2^n+1} \tag{2}$$

According to [19], operand $|B|_{2^n+1}$ is modified Booth encoded as,

$$|B|_{2^n+1} = \boldsymbol{b}^{MB}_{\lceil n/2\rceil-1} \boldsymbol{b}^{MB}_{\lceil n/2\rceil-2} \cdots \boldsymbol{b}^{MB}_1 \boldsymbol{b}^{MB}_0 = \left| \sum_{i=0}^{\lceil n/2\rceil-1} \boldsymbol{b}^{MB}_i \cdot 2^{2i} \right|_{2^n+1}$$

where $\boldsymbol{b}^{MB}_i \in \{-2, -1, 0, +1, +2\}$ (3a)

The digits $\boldsymbol{b}^{MB}_i$ are formed as follows:

When n is odd,

$\boldsymbol{b}^{MB}_i = -2b_{2i+1} + b_{2i} + b_{2i-1}$ for $1 \le i \le \lceil n/2\rceil - 1$ and

$$\boldsymbol{b}^{MB}_0 = b_0 \vee b_n - 2(b_1 \vee b_n) \tag{3b}$$

while for n even,

$\boldsymbol{b}^{MB}_i = -2b_{2i+1} + b_{2i} + b_{2i-1}$ for $2 \le i \le \lceil n/2\rceil - 1$,

$\boldsymbol{b}^{MB}_1 = -2b_3 + b_2 + b_1 \cdot \overline{b_n \vee b_{n-1}}$ and

$$\boldsymbol{b}^{MB}_0 = -2(b_n \vee b_{n-1}) \oplus b_1 + b_0 + b_n \vee b_{n-1} \tag{3c}$$

Then for the product Q we get that

$$Q = \left| \sum_{i=0}^{\lceil n/2\rceil-1} \right| (A_{n-1:0} + 1) \boldsymbol{b}^{MB}_i \cdot 2^{2i}|_{2^n+1} - a_n B - B|_{2^n+1} \tag{4}$$

The computation of the terms $|(A_{n-1:0}+1)\boldsymbol{b}^{MB}_i 2^{2i}|_{2^n+1}$ has been presented in [23] and is depicted in Table 1 as the PP_i terms, each one requiring a constant correction equal to 1, except those with a zero value requiring a correction of $2^{2i}+1$. The above is clearly illustrated by the next relation

Table 1
Formation of the partial products.

$\boldsymbol{b}^{MB}_i$	Meaning	$\boldsymbol{PP_i}$	$\boldsymbol{cor_i}$
0	0	$\underbrace{11\cdots1}_{n-2i}\ \underbrace{00\cdots0}_{2i}$	$1+2^{2i}$
+1	$\|(A_{n-1:0}+1)2^{2i}\|_{2^n+1}$	$\underbrace{a_{n-1-2i}a_{n-2-2i}\cdots a_0}_{n-2i}\ \underbrace{\overline{a}_{n-1}\cdots\overline{a}_{n+1-2i}\overline{a}_{n-2i}}_{2i}$	+1
−1	$\|-(A_{n-1:0}+1)2^{2i}\|_{2^n+1}$	$\underbrace{\overline{a}_{n-1-2i}\overline{a}_{n-2-2i}..\overline{a}_0}_{n-2i}\ \underbrace{a_{n-1}..a_{n+1-2i}a_{n-2i}}_{2i}$	+1
+2	$\|(A_{n-1:0}+1)2^{2i+1}\|_{2^n+1}$	$\underbrace{a_{n-2-2i}\cdots a_1 a_0}_{n-2i-1}\ \underbrace{\overline{a}_{n-1}\cdots\overline{a}_{n-2i}\overline{a}_{n-1-2i}}_{2i+1}$	+1
−2	$\|-(A_{n-1:0}+1)2^{2i+1}\|_{2^n+1}$	$\underbrace{\overline{a}_{n-2-2i}\cdots\overline{a}_1\overline{a}_0}_{n-2i-1}\ \underbrace{a_{n-1}\cdots a_{n-2i}a_{n-2i-1}}_{2i+1}$	+1

$$|(A_{n-1:0}+1)\boldsymbol{b}^{MB}_i 2^{2i}|_{2^n+1} = PP_i + 1 + z_{2i} 2^{2i} \tag{5}$$

where each term z_{2i} is equal to 1 when its corresponding MB digit $\boldsymbol{b}^{MB}_i$ is equal to zero. Therefore, the product Q is computed as follows:

$$Q = \left| \sum_{i=0}^{\lceil n/2\rceil-1} PP_i + \lceil n/2\rceil + Z_n - a_n B - B \right|_{2^n+1} \tag{6}$$

where $Z_n = \sum_{i=0}^{\lceil n/2\rceil-1} z_{2i} 2^{2i}$. Obviously operand Z_n is of the form $z_{n-1}0\cdots0z_2 0 z_0$ for n odd and of the form $0z_{n-2}0\cdots0z_2 0 z_0$ for n even. Terms z_{2i} are derived through a NOR gate by slightly modifying the original Booth Encoding block as is shown in Fig. 3.

From relation (6) we get that

$$Q = \left| \sum_{i=0}^{\lceil n/2\rceil-1} PP_i + \lceil n/2\rceil + Z_n - a_n(b_n 2^n + B_{n-1:0}) - b_n 2^n - B_{n-1:0} \right|_{2^n+1} \text{ or}$$

$$Q = \left| \sum_{i=0}^{\lceil n/2\rceil-1} PP_i + \lceil n/2\rceil + Z_n + b_n + a_n b_n - (a_n B_{n-1:0} + B_{n-1:0}) \right|_{2^n+1} \tag{7}$$

Let $B_l = |a_n \cdot b_n - (a_n B_{n-1:0} + B_{n-1:0})|_{2^n+1}$. For $a_n = 0$ we have

$$B_l = |-B_{n-1:0}|_{2^n+1} = |-(b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + \cdots + b_1 2 + b_0)|_{2^n+1} \tag{8}$$

while for $a_n = 1$

$$B_l = |b_n - 2B_{n-1:0}|_{2^n+1} = |b_n - (b_{n-1}2^n + b_{n-2}2^{n-1} + \cdots + b_0 2)|_{2^n+1}$$
$$= |b_n + b_{n-1} - (b_{n-2}2^{n-1} + \ldots + b_0 2)|_{2^n+1} \text{ or}$$
$$B_l = |b_n \vee b_{n-1} - (b_{n-2}2^{n-1} + \cdots + b_0 2 + 0)|_{2^n+1} \tag{9}$$

Relations (8) and (9) are unified to the following:

$$B_l = |a_n(b_n \vee b_{n-1}) - \left\{(\overline{a}_n b_{n-1} \vee a_n b_{n-2})2^{n-1} + \cdots + (\overline{a}_n b_1 \vee a_n b_0)2 + \overline{a}_n b_0\right\}|_{2^n+1}$$

or $B_l = |a_n(b_n \vee b_{n-1}) - B_L|_{2^n+1}$, where operand

$$B_L = (\overline{a}_n b_{n-1} \vee a_n b_{n-2})(\overline{a}_n b_{n-2} \vee a_n b_{n-3}) \cdots (\overline{a}_n b_1 \vee a_n b_0)\overline{a}_n b_0 \tag{10}$$

That is,

$$Q = \left| \sum_{i=0}^{\lceil n/2\rceil-1} PP_i + Z_n + \lceil n/2\rceil + b_n + a_n(b_n \vee b_{n-1}) - B_L \right|_{2^n+1}$$

Since for the n-bit operand B_L it holds that $|-B_L|_{2^n+1} = |\overline{B}_L + 2|_{2^n+1}$ we get

$$Q = \left| \sum_{i=0}^{\lceil n/2\rceil-1} PP_i + Z_n + \lceil n/2\rceil + b_n + a_n(b_n \vee b_{n-1}) + \overline{B}_L + 2 \right|_{2^n+1} \tag{11}$$

For the case $b_n = 1$, $B = 100\cdots00$ and $\boldsymbol{b}^{MB}_1 = 0$, then according to Table 1 the partial product $PP_1 = 11\cdots100$. Consequently, the addition of the term b_n can be realized by ORing it with $pp_{1,0}$