Modified Booth Modulo $2^n - 1$ Multipliers

C. Efstathiou, H.T. Vergos, *Member*, *IEEE*, and D. Nikolos, *Member*, *IEEE*

Abstract—$2^n - 1$ is one of the most commonly used moduli in Residue Number Systems. In this paper, we propose a new method for designing modified Booth modulo $2^n - 1$ multipliers, which, for even values of n, require one less partial product than the already known. CMOS implementations reveal that the proposed multipliers compared to earlier solutions offer savings up to 28.7 percent and up to 29.3 percent in the implementation area and execution delay, respectively.

Index Terms—Residue Number System, Mersenne arithmetic, one's complement arithmetic, Booth multipliers, VLSI design.

1 Introduction

A Residue Number System (RNS) is characterized by a set, say $\{m_1, m_2, \ldots, m_L\}$, of L moduli that are pairwise relatively prime [1]. An integer X, with $0 \leq X < M$, where $M = m_1 \times m_2 \times \cdots \times m_L$, has a unique representation in the RNS given by the set of residues $X = \{x_1, x_2, \ldots, x_L\}$, where $x_i = X \bmod m_i$. A two-operand RNS operation, say $\diamond$, is defined as

$$\{z_1, z_2, \ldots, z_L\} = \{x_1, x_2, \ldots, x_L\} \diamond \{y_1, y_2, \ldots, y_L\},$$

where $z_i = (x_i \diamond y_i) \bmod m_i$ and $\diamond$ can be modular addition, subtraction, or multiplication. That is, in an RNS, arithmetic operations are performed in parallel units, each one handling small residues instead of a single unit that handles large numbers.

Moduli choices of the forms $\{2^n, 2^n - 1, 2^n + 1\}$ and $\{2^n, 2^n - 1, 2^{n-1} - 1\}$ have received significant attention because they offer very efficient circuits when considering the area $\times$ time^2 product [2] and efficient converters from and to the binary system [3], [4]. Therefore, designing efficient modulo $2^n - 1$ multipliers is an interesting issue.

The most efficient modulo $2^n - 1$ multipliers based on the use of lookup tables were proposed in [5]. However, due to the exponential growth of ROM sizes with respect to the size of the modulus, implementations based solely on combinational hardware are more suitable for medium and large moduli. In [6], modulo $2^n - 1$ multipliers based on Wallace trees were introduced. The number of partial products in [6] is proportional to the operand length. To overcome this problem, modified Booth modulo $2^n - 1$ multipliers were proposed in [7].

In this paper, we give a new design method for modified Booth modulo $2^n - 1$ multipliers, which, for even values of n, leads to one less product compared to the design in [7]. Static CMOS implementations show that, when n is even, the proposed multipliers offer significant savings in physical area and execution delay against those in [6], [7]. Modulo $2^n - 1$ multipliers based both on Carry Save Adder (CSA) arrays and Wallace tree partial product reduction are examined in this work.

The rest of the paper is organized as follows: Preliminaries are given in Section 2, while the proposed modified Booth modulo $2^n - 1$ multipliers are derived in Section 3. Implementation analysis and results verifying the effectiveness of the proposed architecture are presented in Section 4. Our conclusions are drawn in the last section.

- *C. Efstathiou is with the Informatics Department, TEI of Athens, Ag. Spyridonos St., 12210, Egaleo, Athens, Greece. E-mail: cefsta@teiath.gr.*
- *H.T. Vergos and D. Nikolos are with the Computer Engineering and Informatics Department, Patras University, 26500, Greece and with the Computer Technology Institute, 3 Kolokotroni St., 26221 Patras, Greece. E-mail: vergos@ceid.upatras.gr, nikolosd@cti.gr.*

Manuscript received 5 Dec. 2002; revised 3 Apr. 2003; accepted 5 Aug. 2003. For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 117893.

2 Preliminaries

Hereafter, we adopt the widely used 2-bit recoding form of the modified Booth algorithm for the multiplication of $A \times B$. In this algorithm, successive overlapping triplets of the bits of multiplier B are examined and encoded, each one by a Booth encoder block, as an element of the set $\{-2, -1, 0, +1, +2\}$. Several implementations can be devised for the encoder block, depending on the adoption of a 4 or a 3-bit bus for encoding the five elements of the set. In this work, we adopt the 3-bit bus approach since it led to faster implementations. Fig. 1a and Fig. 1b present the truth table and the implementation adopted for the Booth encoder, respectively. The encoded information produced by each encoder block, along with the multiplicand A, is used for forming a partial product. Every bit of each partial product is produced by a Booth selector block. Fig. 1c and Fig. 1d present the truth table and the implementation adopted for the Booth selector, respectively (the notation $\overline{x}$ is used to denote the complement of x).

The resulting partial products are then reduced to two by successive additions of their bits with equal weight, 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. The reduction of the partial products can be implemented by various architectures; CSA arrays and adder trees (Wallace trees) are some widely used solutions. Compressors of higher degree than a full-adder, which can be considered as a $(3, 2)$ compressor, may also be used. The two operands produced are added in a parallel modulo $2^n - 1$ adder [8], [9] that computes the final result.

3 The Proposed Modified Booth Modulo $2^n - 1$ Multipliers

In this section, we describe the suggested architecture for a modified Booth modulo $2^n - 1$ multiplier. Suppose that $A = a_{n-1}a_{n-2}\cdots a_1a_0$ and $B = b_{n-1}b_{n-2}\cdots b_1b_0$ are two n-bit numbers in modulo $2^n - 1$ representation, where A is the multiplicand and B the multiplier. Let $|A|_x$ denote the modulo x residue of A.

In order to reduce the number of the partial products of the multiplication, the multiplier B can be expressed as:

$$B = \sum_{i=0}^{n-1} b_i 2^i = (b_0 - 2b_1) + \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} (b_{2i-1} + b_{2i} - 2b_{2i+1}) 2^{2i},$$

where $b_j = 0$ for $j \geq n$ and $\lfloor k \rfloor$ denotes the largest integer smaller than or equal to k.

We distinguish the following two cases for n:

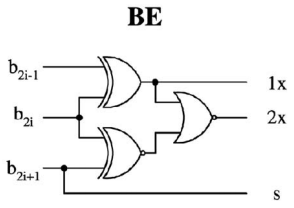
- n is even:

$$B = b_{n-1}2^n + (b_0 - 2b_1) + \sum_{i=1}^{\frac{n}{2}-1} (b_{2i-1} + b_{2i} - 2b_{2i+1}) 2^{2i}.$$

Taking into account that $B = |B|_{2^n-1}$, we get:

b_{2i+1}	b_{2i}	b_{2i-1}	s	2x	1x
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	0	1
1	1	0	1	0	1
1	1	1	1	0	0

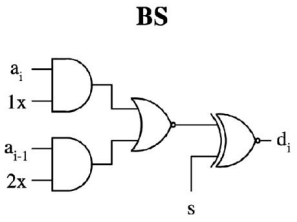
(a)



(b)

s	2x	1x	d_i
0	0	0	0
0	0	1	a_i
0	1	0	a_{i-1}
1	1	0	$\overline{a_{i-1}}$
1	0	1	$\overline{a_i}$
1	0	0	1

(c)



(d)

Fig. 1. Truth tables and implementations for the Booth encoder (a), (b) and the Booth selector blocks (c), (d).

$$\begin{aligned} B = |B|_{2^n-1} &= |b_{n-1}2^n + (b_0 - 2b_1) \\ &\quad + \sum_{i=1}^{\frac{n}{2}-1}(b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i}\Big|_{2^n-1} \\ &= |(b_{n-1} + b_0 - 2b_1) \\ &\quad + \sum_{i=1}^{\frac{n}{2}-1}(b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i}\Big|_{2^n-1}. \end{aligned}$$

Then, setting $b_{-1} = b_{n-1}$, we get:

$$B = \left|\sum_{i=0}^{\frac{n}{2}-1}(b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i}\right|_{2^n-1}. \quad (1)$$

Relation (1) indicates that a single Booth encoder can be utilized for handling the inputs b_1, b_0, and b_{n-1}. In [7], two different Booth encoder blocks are used for handling the same inputs: one that receives as inputs the signals b_1, b_0, and 0 and another that receives the signals 0, 0, and b_{n-1} as inputs. As a result, our architecture requires one less partial product for even values of n. This translates into hardware savings of one Booth encoder, n Booth selectors, and n full-adder blocks, along with their interconnections.

- n is odd:

$$B = \sum_{i=0}^{\frac{n+1}{2}-1}(b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i}, \quad (2)$$

where $b_{-1} = b_n = 0$.

We can combine (1) and (2) into

$$B = \left|\sum_{i=0}^{\lfloor\frac{n+1}{2}\rfloor-1}(b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i}\right|_{2^n-1},$$

where $b_{-1} = b_{n-1}$ for even values of n and $b_{-1} = b_n = 0$ for odd values of n.

The value of the product $A \times B$ modulo $2^n - 1$ can then be expressed as:

$$\begin{aligned} |AB|_{2^n-1} &= \left|A\sum_{i=0}^{\lfloor\frac{n+1}{2}\rfloor-1}(b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i}\right|_{2^n-1} \\ &= \left|\sum_{i=0}^{\lfloor\frac{n+1}{2}\rfloor-1}\left|A(b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i}\right|_{2^n-1}\right|_{2^n-1} \\ &= \left|\sum_{i=0}^{\lfloor\frac{n+1}{2}\rfloor-1} PP_i\right|_{2^n-1}, \end{aligned} \quad (3)$$

where $PP_i = \left|A(b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i}\right|_{2^n-1}$.

Since the terms $b_{e,i} = (b_{2i-1} + b_{2i} - 2b_{2i+1})$ can take values in the set $\{-2, -1, 0, +1, +2\}$, $b_{e,i}2^{2i} = s2^j$, where:

$$s = \begin{cases} +1, & \text{if } b_{e,i} = +1, +2 \\ 0, & \text{if } b_{e,i} = 0 \\ -1, & \text{if } b_{e,i} = -1, -2 \end{cases} \quad \text{and}$$

$$j = \begin{cases} 0, & \text{if } b_{e,i} = 0 \\ 2i, & \text{if } b_{e,i} = +1, -1 \\ 2i+1, & \text{if } b_{e,i} = +2, -2. \end{cases}$$

Substituting this in (3), we get that $PP_i = |sA2^j|_{2^n-1}$.

Using the following lemma, we can express the partial products $PP_i = |sA2^j|_{2^n-1}$, in n bits, directly from the bits of the multiplicand A.

Lemma 1. *Let $A = a_{n-1}a_{n-2}\cdots a_1a_0$ be a number in $[0, 2^n - 1)$. Then,*

$$|sA2^j|_{2^n-1} = \begin{cases} a_{n-1-j}a_{n-2-j}\ldots a_0a_{n-1}a_{n-2}\ldots a_{n-j}, & \text{if } s = +1 \\ \overline{a_{n-1-j}}\,\overline{a_{n-2-j}}\ldots\overline{a_0}\,\overline{a_{n-1}}\,\overline{a_{n-2}}\ldots\overline{a_{n-j}}, & \text{if } s = -1 \\ \underbrace{000\ldots00}_{n \text{ bits}}, & \text{if } s = 0. \end{cases}$$

TABLE 1
Formation of the Partial Products

b_{2i+1}	b_{2i}	b_{2i-1}	Meaning	PP_i
0	0	0	0	$000\cdots000$ or $111\cdots111$ (double representation of zero)
0	0	1	$\lvert A2^{2i}\rvert_{2^n-1}$	$a_{n-1-2i}a_{n-2-2i}\cdots a_0a_{n-1}a_{n-2}\cdots a_{n-2i}$
0	1	0	$\lvert A2^{2i}\rvert_{2^n-1}$	$a_{n-1-2i}a_{n-2-2i}\cdots a_0a_{n-1}a_{n-2}\cdots a_{n-2i}$
0	1	1	$\lvert A2^{2i+1}\rvert_{2^n-1}$	$a_{n-1-(2i+1)}a_{n-2-(2i+1)}\cdots a_0a_{n-1}a_{n-2}\cdots a_{n-(2i+1)}$
1	0	0	$\lvert -A2^{2i+1}\rvert_{2^n-1}$	$\overline{a_{n-1-(2i+1)}}\,\overline{a_{n-2-(2i+1)}}\cdots\overline{a_0}\,\overline{a_{n-1}}\,\overline{a_{n-2}}\cdots\overline{a_{n-(2i+1)}}$
1	0	1	$\lvert -A2^{2i}\rvert_{2^n-1}$	$\overline{a_{n-1-2i}}\,\overline{a_{n-2-2i}}\cdots\overline{a_0}\,\overline{a_{n-1}}\,\overline{a_{n-2}}\cdots\overline{a_{n-2i}}$
1	1	0	$\lvert -A2^{2i}\rvert_{2^n-1}$	$\overline{a_{n-1-2i}}\,\overline{a_{n-2-2i}}\cdots\overline{a_0}\,\overline{a_{n-1}}\,\overline{a_{n-2}}\cdots\overline{a_{n-2i}}$
1	1	1	0	$000\cdots000$ or $111\cdots111$ (double representation of zero)

Proof.

$$\begin{aligned}
A2^j &= \left(\sum_{i=0}^{n-1} a_i 2^i\right)2^j = \left(\sum_{i=0}^{n-1-j} a_i 2^i\right)2^j + \left(\sum_{i=n-j}^{n-1} a_i 2^i\right)2^j \\
&= \left(\sum_{i=0}^{n-1-j} a_i 2^i\right)2^j + \left(\sum_{i=0}^{j-1} a_{n-1-i}2^{j-1-i}\right)2^n \\
&= \left(\sum_{i=0}^{n-1-j} a_i 2^i\right)2^j + \left(\sum_{i=0}^{j-1} a_{n-1-i}2^{j-1-i}\right) \\
&\quad + \left(\sum_{i=0}^{j-1} a_{n-1-i}2^{j-1-i}\right)(2^n-1) \\
&= (a_{n-1-j}a_{n-2-j}\ldots a_0a_{n-1}a_{n-2}\ldots a_{n-j}) \\
&\quad + (a_{n-1}a_{n-2}\ldots a_{n-j})(2^n-1).
\end{aligned} \tag{4}$$

We distinguish the following three cases:

- $s=+1$. Then, using (4), we get:

$$\begin{aligned}
\left|sA2^j\right|_{2^n-1} &= \left|a_{n-1-j}a_{n-2-j}\ldots a_0a_{n-1}a_{n-2}\ldots a_{n-j}\right|_{2^n-1} \\
&= a_{n-1-j}a_{n-2-j}\ldots a_0a_{n-1}a_{n-2}\ldots a_{n-j}.
\end{aligned}$$

- $s=-1$. Taking into account that $|-Y|_x = |x-Y|_x$, from (4), we get:

$$\begin{aligned}
\left|sA2^j\right|_{2^n-1} &= |(2^n-1) \\
&\quad -(a_{n-1-j}a_{n-2-j}\ldots a_0a_{n-1}a_{n-2}\ldots a_{n-j})\big|_{2^n-1} \\
&= \left|\overline{a_{n-1-j}}\,\overline{a_{n-2-j}}\ \ldots\overline{a_0}\,\overline{a_{n-1}}\,\overline{a_{n-2}}\ldots\overline{a_{n-j}}\right|_{2^n-1} \\
&= \overline{a_{n-1-j}}\,\overline{a_{n-2-j}}\ \ldots\overline{a_0}\,\overline{a_{n-1}}\,\overline{a_{n-2}}\ldots\overline{a_{n-j}}.
\end{aligned}$$

- $s=0$. In this case, $|sA2^j|_{2^n-1} = 0 = \underbrace{000\ldots00}_{n \text{ bits}}$. □

Lemma 1 indicates that the partial products can be expressed according to Table 1. According to the first and the last lines of Table 1, an all 1s input will be treated as a zero input, although, in pure modulo 2^n-1 arithmetic, such an operand is not applicable.

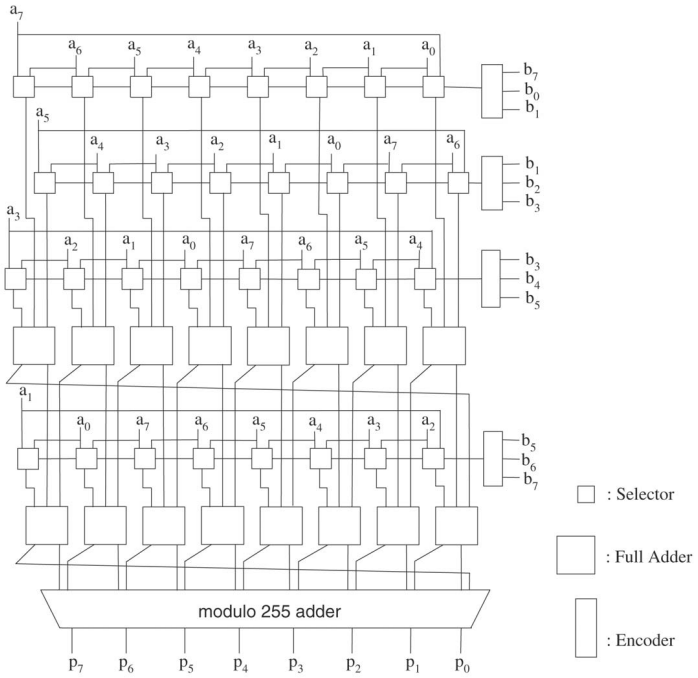


Fig. 2. Proposed modified Booth modulo 255 multiplier.

TABLE 2
Area Comparisons

n	A [6]	A [7]	$A_{Proposed}$	Savings % over [6]	Savings % over [7]
4	112	143	90	19.6	37.1
8	504	497	396	21.4	20.3
16	2080	1805	1608	22.7	10.9
32	8352	6773	6384	23.6	5.7

However, depending on the final adder choice (see [9]), the multiplier can be forbidden from producing the all 1s output.

Fig. 2 presents the proposed design for a modulo $2^8 - 1$ multiplier. As mentioned earlier, a single Booth encoder is used for inputs $\{b_1, b_0, b_7\}$ in contrast to the design in [7], where two different Booth encoder blocks for handling the same inputs are used; one for $\{b_1, b_0, 0\}$ and another for $\{0, 0, b_7\}$.

4 Comparisons

In this section, we compare the proposed multipliers against the multiplier designs of [6] and [7] that are considered the most efficient in the literature.

Let A_{FA} denote the area of a full adder (FA) and T_{FA} its delay, A_{BE} the area of a Booth encoder (BE) and T_{BE} its delay, A_{BS} the area of a Booth selector (BS) and T_{BS} its delay, and A_{PA_n} the area of a modulo $2^n - 1$ adder ($\mathrm{PA}(n)$) and T_{PA_n} its delay. Further, let $k(x)$ denote the depth in FAs of each Wallace tree when adding x partial products in modulo $2^n - 1$ arithmetic. $k(x)$ is equal to 0, 1, 2, 3, 4, 4, 6, 6, and 8 when x is 2, 3, 4, 5, 8, 9, 16, 17, and 32, respectively [10]. The number of FAs required in each tree is equal to $x - 2$.

The multipliers presented in [6] require n^2 gates for forming the partial products. For reducing the n partial products in two summands, either $(n-2)$ CSA stages [7], each consisting of n FAs, or n Wallace trees, each adding n bits of equal weight, can be used. The result is obtained by a final $\mathrm{PA}(n)$. Thus, their total area when either a CSA array or Wallace trees are used is equal to $\mathrm{A}[6] = n^2 + n(n-2)A_{FA} + A_{PA_n}$. Their delay can be modeled by $\mathrm{T}[6]_{CSA} = 1 + (n-2)T_{FA} + T_{PA_n}$ when a CSA array is used and by $\mathrm{T}[6]_{WAL} = 1 + k(n)T_{FA} + T_{PA_n}$ when Wallace trees are used.

The multipliers in [7] produce $(\lfloor \frac{n}{2} \rfloor + 1)$ partial products utilizing $(\lfloor \frac{n}{2} \rfloor + 1)$ BE and $n(\lfloor \frac{n}{2} \rfloor + 1)$ BS blocks. These partial products are reduced to two summands by either $(\lfloor \frac{n}{2} \rfloor - 1)$ CSA stages, each consisting of n FAs, or of n Wallace trees, each adding $\lfloor \frac{n}{2} \rfloor + 1$ bits. The two summands are finally added using a $\mathrm{PA}(n)$. The total area occupied when either a CSA array or Wallace trees are used is equal to

$$\mathrm{A}[7] = \left(\left\lfloor \frac{n}{2} \right\rfloor + 1\right)A_{BE} + n\left(\left\lfloor \frac{n}{2} \right\rfloor + 1\right)A_{BS} + n\left(\left\lfloor \frac{n}{2} \right\rfloor - 1\right)A_{FA} + A_{PA_n}.$$

Their delay can be modeled by $\mathrm{T}[7]_{CSA} = T_{BE} + T_{BS} + (\lfloor \frac{n}{2} \rfloor - 1)T_{FA} + T_{PA_n}$ when a CSA array is used and by $\mathrm{T}[7]_{WAL} = T_{BE} + T_{BS} + k(\lfloor \frac{n}{2} \rfloor + 1)T_{FA} + T_{PA_n}$ when Wallace trees are used.

Finally, the proposed design requires an area equal to $\mathrm{A}_{Proposed} = \lceil \frac{n}{2} \rceil A_{BE} + n\lceil \frac{n}{2} \rceil A_{BS} + n(\lceil \frac{n}{2} \rceil - 2)A_{FA} + A_{PA_n}$ when either a CSA array or Wallace trees are used. The delay of the proposed multipliers is equal to $\mathrm{T}_{Proposed,CSA} = T_{BE} + T_{BS} + (\lceil \frac{n}{2} \rceil - 2)T_{FA} + T_{PA_n}$ when a CSA array is used and equal to $\mathrm{T}_{Proposed,WAL} = T_{BE} + T_{BS} + k(\lceil \frac{n}{2} \rceil)T_{FA} + T_{PA_n}$ when Wallace trees are used.

TABLE 3
Delay Comparisons

CSA based designs					
n	$\mathrm{T}[6]_{CSA}$	$\mathrm{T}[7]_{CSA}$	$\mathrm{T}_{Proposed,CSA}$	Savings % over [6]	Savings % over [7]
4	16	18	14	12.5	22.2
8	34	28	24	29.4	14.3
16	68	46	42	38.2	8.7
32	134	80	76	43.3	5.0
Wallace trees based designs					
n	$\mathrm{T}[6]_{WAL}$	$\mathrm{T}[7]_{WAL}$	$\mathrm{T}_{Proposed,WAL}$	Savings % over [6]	Savings % over [7]
4	16	18	14	12.5	22.2
8	26	28	24	7.7	14.3
16	36	34	34	5.6	0.0
32	46	44	44	4.4	0.0

TABLE 4
Implementation Results for Modulo $2^n - 1$ Booth Multipliers with Wallace Trees Partial Products Reduction

	Multipliers of [6]		**Multipliers of [7]**		**Proposed Multipliers**	
n	*Area (mils^2)*	*Delay (ns)*	*Area (mils^2)*	*Delay (ns)*	*Area (mils^2)*	*Delay (ns)*
4	286.3	6.02	289.9	6.07	219.1	4.29
8	1092.0	9.65	1243.3	9.94	1089.5	9.13
16	3915.6	14.19	4262.7	14.34	3835.7	13.27
32	11649.9	19.99	10712.1	19.73	10290.4	18.82

Based on the unit-gate model [11], we get the following approximations:

- $A_{BE} = 5$ equivalent gates, $T_{BE} = 3$ time units (see Fig. 1b),
- $A_{BS} = 5$ equivalent gates, $T_{BS} = 4$ time units (see Fig. 1d),
- $A_{FA} = 7$ equivalent gates, $T_{FA} = 4$ time units, and
- $A_{PA_n} = 3n \log n + 4n$ equivalent gates, $T_{PA_n} = 2 \log n + 3$ time units [9].

The area savings of the proposed design against the architectures of [6] and [7] is shown in Table 2 for $n = 4, 8, 16$, and 32. The delay savings of the proposed design against the architectures in [6] and [7] when either a CSA array or a Wallace tree partial products reduction scheme is used are presented in Table 3.

The proposed designs are up to 23.6 percent more area efficient than the multipliers proposed in [6], with an average area savings of 21.8 percent. Compared against the multipliers of [7], the proposed multipliers offer smaller implementations by 18.5 percent on the average and by more than 10 percent when $n \leq 16$. In parallel, the proposed multipliers lead to up to 43.3 percent and 22.2 percent faster implementations than the architectures in [6] and [7], respectively, when a CSA array is used. The corresponding savings are up to 12.5 percent and 22.2 percent when Wallace trees are used. On average, the proposed multipliers are faster by the multipliers in [6] and [7] by 30.9 percent and 12.6 percent, respectively, when a CSA is used and by 7.5 percent and 9.1 percent, respectively, when Wallace trees are used.

The above comparisons are only an approximation since they ignore routing as well as fan-in and fan-out requirements. For a more accurate comparison, we implemented our proposed modulo $2^n - 1$ multipliers and compared them against those of [6] and [7] in static CMOS VLSI technology. We first described each multiplier in a hardware description language and then mapped them to the AMS® CUB® implementation technology ($0.6\mu m$, 2-metal layer, 5.0 V), using the Synopsys® Design Compiler® tool. During synthesis, the netlists were optimized for speed and, as a secondary target, the tool was instructed to try to recover as much area as possible. All the presented results associated with execution latency assume worst-case process parameters and are measured in nanoseconds (ns), whereas we use square milimeters of an inch (mils^2) as a metric for area-related results.

We implemented both the proposed modulo $2^n - 1$ multipliers and those in [6] and [7] for $n = 4, 8, 16$, and 32 bits, using Wallace trees partial products reduction, since this reduction scheme, according to Tables 2 and 3, leads to faster implementations, without increasing the occupied physical area. In all designs, we used the fast modulo $2^n - 1$ parallel-prefix adder proposed in [9] as the final stage adder. The obtained results are listed in Table 4. The results indicate that the proposed multipliers are up to 23.5 percent more area efficient than the multipliers in [6], while being up to 28.7 percent faster. The average area and delay savings in the examined operand width range are 9.4 percent and 11.6 percent, respectively. The proposed multipliers are also up to 28.7 percent more area efficient than the multipliers in [7] while being up to 29.3 percent faster. In this case, the average area and delay savings are 12.7 percent and 12.4 percent, respectively.

5 Conclusions

In this paper, we have presented a new architecture for modified Booth modulo $2^n - 1$ multipliers. The resulting multipliers compare favorably, with respect to speed and implementation area, against the known modulo $2^n - 1$ multipliers.

Acknowledgments

The authors would like to acknowledge the reviewers' valuable comments on the earlier versions of this paper.

References

[1] M.A. Soderstrand et al., *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*. IEEE Press, 1986.
[2] V. Paliouras and T. Stouraitis, "Novel High-Radix Residue Number System Multipliers and Adders," *Proc. IEEE Int'l Symp. Circuits and Systems*, pp. 451-454, 1999.
[3] W. Wang et al., "A High-Speed Residue-to-Binary Converter for Three-Moduli $(2^k, 2^k - 1, 2^{k-1} - 1)$ RNS and a Scheme for Its VLSI Implementation," *IEEE Trans. Circuits and Systems II*, vol. 47, no. 12, pp. 1576-1581, 2000.
[4] Y. Wang et al., "Adder Based Residue to Binary Number Converters for $(2^n - 1, 2^n, 2^n + 1)$," *IEEE Trans. Signal Processing*, vol. 50, no. 7, pp. 1772-1779, 2002.
[5] A. Skavantzos and P.B. Rao, "New Multipliers Modulo $2^n - 1$," *IEEE Trans. Computers*, vol. 41, no. 8, pp. 957-961, Aug. 1992.
[6] Z. Wang, G.A. Jullien, and W.C. Miller, "An Algorithm for Multiplication Modulo $(2^n - 1)$," *Proc. 39th Midwest Symp. Circuits and Systems*, pp. 1301-1304, 1997.
[7] R. Zimmerman, "Efficient VLSI Implementation of Modulo $(2^n \pm 1)$ Addition and Multiplication," *Proc. 14th IEEE Symp. Computer Arithmetic*, pp. 158-167, Apr. 1999.
[8] C. Efstathiou, D. Nikolos, and J. Kalamatianos, "Area-Time Efficient Modulo $2^n - 1$ Adder Design," *IEEE Trans. Circuits and Systems II*, vol. 41, no. 7, pp. 463-467, 1994.
[9] L. Kalampoukas et al., "High-Speed Parallel-Prefix Modulo $2^n - 1$ Adders," *IEEE Trans. Computers*, vol. 49, no. 7, pp. 673-680, July 2000.
[10] I. Koren, *Computer Arithmetic Algorithms*, second ed. Natick, Mass.: A.K. Peters, 2002.
[11] A. Tyagi, "A Reduced-Area Scheme for Carry-Select Adders," *IEEE Trans. Computers*, vol. 42, no. 10, pp. 1163-1170, Oct. 1993.