EFFICIENT MODULO 2^N+1 TREE MULTIPLIERS FOR DIMINISHED-1 OPERANDS

C. Efstathiou[1], H. T. Vergos[2,3], G. Dimitrakopoulos[2], & D. Nikolos[2,3]

[1] Dept. of Informatics, TEI of Athens, Ag. Spyridonos Str., 12210 Egaleo, Athens, Greece
[2] Computer Engineering & Informatics Dept., University of Patras, 26500 Rio, Greece
[3] Computer Technology Institute, 3 Kolokotroni Str., 26221 Patras, Greece
E-mails : cefsta@teiath.gr, vergos@ceid.upatras.gr, dimitrak@ceid.upatras.gr, nikolosd@cti.gr

ABSTRACT

In this work we propose a new method for designing modulo 2^n+1 multipliers for diminished-1 operands. Our multipliers compared to the already known tree architecture offer enhanced operation speed for the majority of n values, with similar area complexities. They also have very regular structure, and can be pipelined at the full-adder level.

1. INTRODUCTION

Arithmetic modulo *A*, where *A* is a positive integer, has been used extensively in digital computing systems. Especially multiplication modulo 2^n+1 is used in many applications.

Specialized processors, based on the Residue Number System (RNS) [1], [2], is a first application field. RNS based on moduli of the form $\langle 2^n-1,\ 2^n,\ 2^n+1\rangle$ have received significant attention because they offer very efficient circuits in the area x time^2 product sense [3]. Addition or multiplication in such systems is performed using three independent channels, that are a modulo 2^n-1, a modulo 2^n and a modulo 2^n+1 adder or multiplier respectively.

Modulo 2^n+1 multiplication finds also applicability in pseudorandom number generation [4], in cryptography algorithms that use modulo exponentiation [5] and in the Fermat number transform, which is used effectively to compute convolutions without round off errors [10]. In all above applications high-speed data rates can be greatly appreciated; therefore efficient implementati-ons of modulo 2^n+1 multiplication are welcome.

In order to speed up the modulo 2^n+1 arithmetic operations the diminished-1 representation of binary numbers was introduced in [6]. In this representation a number $A,\in\left[0,2^n+1\right)$ is represented by $A_{-1}=A-1$, while zero is handled separately. Since the diminished-1 representation requires *n* bits, the arithmetic circuits are more efficient than their corresponding circuits for regular (non-diminished) operands.

Efficient carry look ahead and totally parallel-prefix modulo 2^n+1 adders for diminished-1 operands, which operate as fast as the corresponding modulo 2^n and 2^n-1 adders have been proposed in [7], while Select-Prefix adders suitable for wide operands have been proposed in [8].

Modulo 2^n+1 multiplier architectures for dimimished-1 operands based on a Wallace tree have been proposed in [9]. The delay of the multiplication for wide operands is improved using the diminished-1 modified Booth modulo 2^n+1 multipliers proposed in [10].

For improving the throughput of the multipliers, synchronizing elements may be inserted, leading to a pipelined design. This technique can be applied effectively to the multipliers proposed in [10] but not to the modified Booth multipliers, since the hardware overhead required is very large. Therefore, the design of more time efficient modulo 2^n+1 multipliers, which can be pipelined efficiently at the full-adder level, is an attractive problem.

In the present work we propose new tree modulo 2^n+1 multiplier architectures, which for the majority of the values of *n*, operate faster than those proposed in [9]. They also offer a very regular structure, maintain low hardware cost and can easily be pipelined at the full adder level.

2. MODULO 2^N+1 MULTIPLIERS

Let A, B be two $(n+1)$-bit numbers, $A,B\in\left[0,2^n+1\right)$, and $A_{-1}=a_{n-1}a_{n-2}\ldots a_1a_0$, $B_{-1}=b_{n-1}b_{n-2}\ldots b_1b_0$ their diminished-1 binary representations. Hereafter, the modulo Y residue of X will be denoted as $\left|X\right|_Y$. If $Q=\left|A\cdot B\right|_{2^n+1}$ is the product of *A*, *B* modulo 2^n+1, its diminished-1 representation is:

$$\begin{aligned}Q_{-1}&=Q-1=\left|(A_{-1}+1)(B_{-1}+1)\right|_{2^n+1}-1=\\&=\left|A_{-1}B_{-1}+A_{-1}+B_{-1}+1\right|_{2^n+1}-1=\\&=\left|A_{-1}B_{-1}+A_{-1}+B_{-1}\right|_{2^n+1}\end{aligned}$$

or $Q_{-1} = \left| |A_{-1}B_{-1}|_{2^n+1} + A_{-1} + B_{-1} \right|_{2^n+1}$.

The product $A_{-1}B_{-1}$ modulo 2^n+1 can be computed as follows:

$$|A_{-1}B_{-1}|_{2^n+1} = \left| \left(\sum_{i=0}^{n-1} a_i 2^i \right) \left(\sum_{j=0}^{n-1} b_j 2^j \right) \right|_{2^n+1} =$$

$$= \left| \sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} a_i b_j 2^{i+j} \right) \right|_{2^n+1} =$$

$$= \left| \sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} \left| a_i b_j 2^{i+j} \right|_{2^n+1} \right) \right|_{2^n+1} =$$

$$= \left| \left(\sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} \left| a_i b_j \left| 2^{i+j} \right|_{2^n+1} \right|_{2^n+1} \right) \right) \right|_{2^n+1} =$$

$$= \left| \sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} \left| a_i b_j (-1)^s 2^{|i+j|_n} \right|_{2^n+1} \right) \right|_{2^n+1},$$

where $s = \begin{cases} 0, if \;\; i+j < n \\ 1, if \;\; i+j \geq n \end{cases}$.

Since, for z in {0, 1}

$|-z|_{2^n+1} = |2^n + 1 - z|_{2^n+1} = |2^n + \overline{z}|_{2^n+1}$,

we have

$$|A_{-1}B_{-1}|_{2^n+1} = \left| \left(\sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} x_{i,j} 2^{|i+j|_n} \right) \right) \right|_{2^n+1}, \text{ where}$$

$$x_{i,j} = \begin{cases} a_i b_j, if \;\; i+j < n \\ \left| 2^n + \overline{a_i b_j} \right|_{2^n+1}, if \;\; i+j \geq n \end{cases} \qquad (1).$$

Relation (1) means that each partial product term $a_i b_j$ with $i+j > n$ is complemented and shifted to the position with weight $2^{|i+j|_n}$ introducing a correction factor $2^n 2^{|i+j|_n}$.

Since the second partial product introduces a correction factor of $2^0 2^n$, the third a correction factor of $(2^0 + 2^1)2^n = (2^3 - 1)2^n$ and so on up to the n-th partial product which introduces a correction factor of $(2^0 + 2^1 + ... + 2^{n-2})2^n = = (2^{n-1} - 1)2^n$.

The total correction factor required is

$$(2(1 + 2 + 2^2 + ... + 2^{n-2}) - (n-1))2^n =$$
$$= ((2^n - 2) - (n-1))2^n =$$
$$= (2^n - n - 1)2^n . \qquad (2)$$

The addition of the derived operands in a modulo 2^n+1 fashion can be performed by either a $(n+1)-$stage Carry Save Adder (CSA) array or a Wallace tree, until a Carry and Sum vector pair is reached.

It is well known that the addition of the partial products produced by the conventional multiplication algorithm is speeded up using a Wallace tree, resulting however to irregular architectures. As we will exemplify later, in our case the Wallace tree is highly regular.

The carries with weight 2^n, generated by the partial products' modulo 2^n+1 addition, according to:

$$\left| c_i 2^n \right|_{2^n+1} = |-c_i|_{2^n+1} = \left| 2^n + \overline{c_i} \right|_{2^n+1},$$

are complemented and added in an end-around carry fashion to the least significant bit position of the next stage. The correction factor introduced in this case is :

$$(n+1)2^n \qquad (3)$$

Therefore, according to relations (2) and (3) the total correction required is

$$COR = \left| (2^n - n - 1)2^n + (n+1)2^n \right|_{2^n+1} =$$
$$= \left| 2^n 2^n \right|_{2^n+1} = 1$$

According to the above analysis, the product Q_{-1} is computed according to the following relation

$$Q_{-1} = \left| \left(\sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} x_{i,j} 2^{|i+j|_n} \right) \right) + A_{-1} + B_{-1} + COR_{-1} \right|_{2^n+1}, \text{ where}$$

$$x_{i,j} = \begin{cases} a_i b_j \;\; if \;\; i+j < n \\ \overline{a_i b_j}, \;\; if \;\; i+j \geq n \end{cases}, \text{ and}$$

COR_{-1} is the diminished-1 representation of the total correction. In modulo 2^n+1 arithmetic COR_{-1} is the n-bit vector $000 \ldots 0$. Therefore, the stage of full adders (FA) accepting this operand can be simplified to a stage of half adders (HA).

The Sum and Carry vector pairs that result from the Wallace tree are finally added using a modulo 2^n+1 adder for diminished-1 operands [7], [8].

The proposed design methodology is exemplified in the following.

Example. For the modulo 17 multiplication according to our methodology the following operands are derived:

2^3	2^2	2^1	2^0
a_3b_0	a_2b_0	a_1b_0	a_0b_0
a_2b_1	a_1b_1	a_0b_1	$\overline{a_3b_1}$
a_1b_2	a_0b_2	$\overline{a_3b_2}$	$\overline{a_2b_2}$
a_0b_3	$\overline{a_3b_3}$	$\overline{a_2b_3}$	$\overline{a_1b_3}$
a_3	a_2	a_1	a_0
b_3	b_2	b_1	b_0
0	0	0	0

An implementation of the modulo 17 multiplier based on a Wallace tree for addition the operands is shown in Figure 1. The small circles on the carry output of the adders represent a complemented carry output. It is evident that the derived tree multiplier has a very regular structure, which can be easily pipelined up to the FA level. The final adder is implemented in a parallel-prefix form [7] and can also be pipelined.

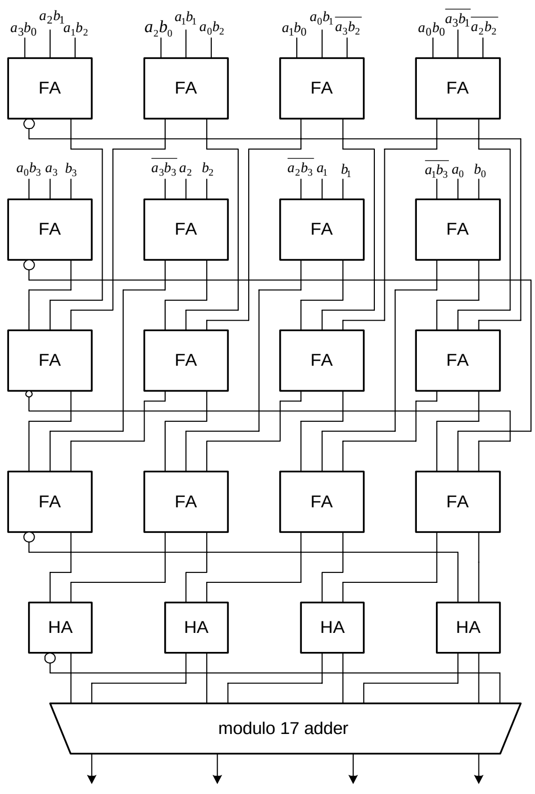

Figure 1. Proposed modulo 17 multiplier for diminished-1 operands

3. COMPARISONS

In this section we compare the proposed multipliers against those of [9] in terms of both delay and area. Our comparisons are based on the commonly used unit-gate model [11]. In this model the 2-input monotonic gates (NAND, AND, etc) count as one equivalent for both area and delay, while an XOR/XNOR gate as 2 equivalents for both area and delay. A full adder has an area and delay complexity of 7 and 4 equivalents respectively, while those of a half adder are 3 and 2 respectively. This model is realistic for the compared designs since they both have limited fanout.

For the most efficient implementation of the multiplication algorithm proposed in [9], the *n* partial products are added by a Wallace tree. The resulting sum and carry vectors are added along with the operand $111...1\bar{z}_{\lceil log_2(n-1)\rceil -1}\ldots\bar{z}_1\bar{z}_0$ in a CSA. $z_{\lceil log_2(n-1)\rceil -1}\cdots z_1 z_0$ is the number of zeros in the *n*-1 most significant bits of the operand B and are computed by a $(n-1,\lceil log_2(n-1)\rceil)$ parallel counter [12].

The above addition operations require $ST_{[9]}(n)=ST(n)+1$ full adder stages, where $ST(n)$ is given in Table 1 taken from [9]. The full adders of the CSA, which have one of their inputs set to logic 1, are simplified in our comparisons in modules implementing the functions $c_i = a_i + b_i$, $s_i = \overline{a_i \oplus b_i}$. Their complexity is considered equivalent to that of the half adder.

Table 1. FA stages in a Wallace tree as a function of added operands

Number of operands N	Number of FA stages ST
4	2
$5 \le N \le 6$	3
$7 \le N \le 9$	4
$10 \le N \le 13$	5
$14 \le N \le 19$	6
$20 \le N \le 28$	7
$29 \le N \le 42$	8
$43 \le N \le 63$	9
$63 \le N \le 94$	10

The final adder of the multipliers proposed in [9], is a modulo 2^n+1 adder with its carry input set to logic 1. We consider that this is implemented as a stage of *n* half adders, followed by a fast modulo 2^n+1 adder [7, 8]. This implementation has a complexity equivalent to the last two stages of the proposed design.

We also note that one of the partial products introduced by the algorithm in [9] is derived using two gate levels. If *n* is a multiple of 3, this introduced additional gate level delay cannot be eliminated, by driving this partial product to the second full adder stage.

According to the above analysis the delay of the modulo multipliers of [9] is

$$T_{[9]}(n) = D + 4ST_{[9]}(n) + T_{PA(n)},$$

where $D = \begin{cases} 3 \text{ if } n \ne 3k \\ 4 \text{ if } n = 3k \end{cases}$, and

$T_{PA(n)}$ is the delay of the final adder.

The delay of the proposed in this work modulo multipliers is easily computed as

$$T_{PROP}(n) = 3 + 4ST_{PROP}(n) + T_{PA(n)},$$

where $ST_{PROP}(n)$ is the number of full adder stages in our design estimated according to Table 1 as $ST_{PROP}(n)$ = $ST(n+2)$. Table 2 gives the full adder stages and the corresponding gate levels of our architecture compared to those of [9].

From the above we conclude that our design is implemented with 5, 4 or 1 gate level less, for the majority of the values of n.

The area complexity of the multipliers in [9] is $A_{[9]} = 8n^2 - 6n + 4\lceil log(n-1)\rceil + A_{c(n-1)} + A_{PA(n)}$, where $A_{c(n-1)}$ is the area of the $(n-1,\lceil log(n-1)\rceil)$ parallel counter and $A_{PA(n)}$ the area of the final adder.

Table 2. Comparison of full adder stages

	[9]		Proposed	
N	**FA stages**	**Gate levels**	**FA stages**	**Gate levels**
4	3	12	3	12
5, 6	4	16	4	16
7	5	20	4	16
8, 9	5	20	5	20
10,11	6	24	5	20
12,13	6	24	6	24
$14 \le n \le 17$	7	28	6	24
18,19	7	28	7	28
$20 \le n \le 26$	8	32	7	28
27,28	8	32	8	32
$29 \le n \le 40$	9	36	8	32
41, 42	9	36	9	36
$43 \le n \le 61$	10	40	9	36
62,63	10	40	10	40
64	11	44	10	40

The area complexity of the proposed design is

$$A_{PROP} = 8n^2 + 3n + A_{PA(n)}$$

According to Table 3, which gives the area of our architecture compared to that of [9] for representative values of n, the proposed design has area complexity close to that of [9].

Table 3. Multiplier area estimation

n	**[9]**	**Proposed**
8	$504+A_{PA(8)}$	$536+A_{PA(8)}$
16	$2045+A_{PA(16)}$	$2096+A_{PA(16)}$
32	$8202+A_{PA(32)}$	$8288+A_{PA(32)}$
64	$32807+A_{PA(64)}$	$32957+A_{PA(64)}$

4. CONCLUSIONS

We have proposed a new methodology for designing diminished-1 modulo 2^n+1 multipliers. Compared to an earlier design, the proposed multipliers are faster in the majority of the values of n. In parallel they maintain low hardware complexity and can be pipelined efficiently at the full-adder level.

REFERENCES

[1] M. A. Sonderstrand, et al., Residue Number System Arithmetic: Modern Applications in Digital Signal Processing, IEEE Press, New York, 1986.

[2] M. A. Bayoumi, et al., "A Look-Up Table VLSI Design Methodology for RNS Structures used in DSP Applications", IEEE Trans. on Circuits and Systems, vol. CAS-34, pp. 604-616, June 1987.

[3] V. Paliouras and T. Stouraitis, "Novel High – Radix Residue Number System Multipliers and Adders", in Proc. of the 1999 IEEE Int. Symposium on Circuits and Systems VLSI (ISCAS '99), Orlando, FL, USA, pp. 451 – 454, 1999.

[4] D. H. Lehmer, in Proc. of the 2nd Symp. on Large – Scale Digital Calculating Machinery, Cambridge, MA : Harvard University Press, pp. 141 – 146, 1951.

[5] R. Zimmermann, et al., "A 177 Mb/s VLSI Implementation of the International Data Encryption Algorithm", IEEE J. of Solid - State Circuits, vol. 29 (3), pp. 303 – 307, March 1994.

[6] L. M. Leibowitz, "A simplified binary arithmetic for the Fermat number transform", IEEE Trans. Acoust. Speech, Signal Processing, vol. ASSP-24, pp. 356-359, 1976.

[7] H. T. Vergos, et al. "Diminished-One Modulo 2^n+1 Adder Design", IEEE Trans. on Computers, vol. 15, no. 12, pp. 1389-1399, Dec. 2002.

[8] C. Efstathiou,et al., "Modulo 2^n+1 Adder Design Using Select-Prefix Blocks", to appear in the IEEE Trans. on Computers.

[9] Z. Wang, et al. "An Efficient Tree Architecture for Modulo 2^n+1 multiplication", J. of VLSI Signal Processing, vol.14, pp. 241-248, 1996.

[10] Y. Ma, "A Simplified Architecture for Modulo (2^n+1) Multiplication", IEEE Trans. on Computers, vol. 47, no. 3, pp. 333-337, March 1998.

[11] R. Zimmermann, "Efficient VLSI implementation of modulo $2^n \pm 1$ addition and multiplication", Proc. 14th Symp. Computer Arithmetic, pp. 158-167, Apr. 1999.

[12] E. E. Swartzlander, "Parallel Counters", IEEE Trans. on Computers, vol. 22, no. 11, pp. 1021-1024, March 1973.