



VLSI Research Group

University of Windsor

*An Efficient Tree Architecture for
Modulo 2^n+1 Multiplication*

Journal of VLSI Signal Processing

Zhongde Wang, G. A. Jullien, and W. C. Miller

An Efficient Tree Architecture for Modulo 2^n+1 Multiplication

Zhongde Wang, G. A. Jullien, and W. C. Miller

Abstract

Modulo $(2^n + 1)$ multiplication plays an important role in the Fermat number transform and residue number systems; the diminished-1 representation of numbers has been found most suitable for representing the elements of the rings. Existing algorithms for modulo $(2^n + 1)$ multiplication either use recursive modulo $(2^n + 1)$ addition, or a regular binary multiplication integrated with the modulo reduction operation. Although most often adopted for large n , this latter approach requires conversions between the diminished-1 and binary representations. In this paper we propose a parallel fine-grained architecture, based on a Wallace tree, for modulo $(2^n + 1)$ multiplication which does not require any conversions; the use of a Wallace tree considerably improves the speed of the multiplier. This new architecture exhibits an extremely modular structure with associated VLSI implementation advantages. The critical path delay and the hardware requirements of the new multiplier are similar to that of a corresponding $n \times n$ bit binary multiplier.

1.0 Introduction

The Fermat number transform has found applications in signal processing [1], and computation over a Fermat Number Field has recently been proposed in a modulus replication architecture [2]. To develop an efficient coding system for Fermat number arithmetic, Leibowitz proposed the diminished-1 coding system [3], which is not only suitable for Fermat numbers, but also for general moduli of the form $(2^n + 1)$. Computations over modulo $(2^n + 1)$ rings, based on these numbers, is also adopted for many residue number system implementations [4][5]. Multiplication modulo $(2^n + 1)$ is an important, but rather difficult operation, and has been a topic of many published papers [3][6][7][8][9] [10][11]. Since the diminished-1 coding is efficient and convenient, most published modulo $(2^n + 1)$ multiplication algorithms use this representation [3][7][8][9]. In his paper, Leibowitz suggested two algorithms for multiplication of diminished-1 coded numbers [3]. Chang et al proposed an improved algorithm for multiplication of diminished-1 coded numbers [7], which requires a conversion to binary representation prior to the multiplication. In the same paper, they also proposed a way to merge the carry correction into the addition process. Benaissa et al [8] made another improvement to avoid the conversion from the diminished-1 representation. Both of these techniques, however, require a large number of n -bit diminished-1 additions. Based on an investigation of several approaches, Curiger et al [10] conclude that, for large n , a modulo $(2^n + 1)$ multiplier implemented by a $(n + 1) \times (n + 1)$ bit binary multiplier together with modulo reduction is the best solution. The multiplier proposed in [11] uses a modification of this scheme with a most significant bit (MSB) detector and a smaller $n \times n$ bit binary multiplier.

In this paper we introduce a new modulo $(2^n + 1)$ multiplier which avoids all drawbacks of the cited multipliers. The large number of n -bit diminished-1 additions in the multipliers of [7] and [8] are replaced by a Wallace tree of full adders, and the length- $2n$ adder for the $n \times n$ bit binary multiplier in [11] is replaced by a length- n diminished-1 adder. In this new architecture no conversion is required, since number representation is always in the

diminished-1 form, and the usual redundancy associated with representing numbers over the Mod $(2^n + 1)$ ring is eliminated.

This paper is organized as follows. In Section 2.0, the diminished-1 representation is briefly reviewed. The new algorithm for modulo $(2^n + 1)$ multiplication is introduced in Section 3.0. Section 5 describes the architecture of the new multiplier, with an illustrative example for $n=8$. Comparisons and the conclusions are presented in Section 5.0 and Section 6.0, respectively.

2.0 Diminished-1 representation

The diminished-1 representation of numbers was proposed by Leibowitz [3], as a convenient and efficient form for modulo $(2^n + 1)$ operations on binary numbers. If we let $d(A)$ be the diminished-1 representation of A , then:

$$d(A) = (A - 1) \text{Mod}(2^n + 1) \quad (1)$$

The advantage of this representation is that zero is uniquely identified by MSB=1, for which case all arithmetic operations are inhibited.

We obtain the following relationships [3] for arithmetic operations within the representation:

$$d(A + B) = d(A) \oplus d(B) = d(A) + d(B) + 1 \text{ Mod}(2^n + 1) \quad (2)$$

$$d(A - B) = d(A) \oplus [-d(B)] = d(A) + \overline{d(B)} + 1 \text{ Mod}(2^n + 1) \quad (3)$$

$$d\left(\sum_{k=1}^n A_k\right) = \sum_{k=1}^n \oplus d(A_k) = \quad (4)$$

$$d(A_1) \oplus \dots \oplus d(A_n) = d(A_1) + \dots + d(A_n) + n - 1 \text{ Mod}(2^n + 1)$$

where \oplus represents addition, $[-x]$, negation in the diminished-1 representation, respectively. $\overline{d(B)}$ represents the one's complement of $d(B)$, and $\sum_{\oplus} d(A_k)$ represents modulo $(2^n + 1)$ summation of diminished-1 numbers.

Since $2^n \equiv -1 \pmod{2^n + 1}$ the residue operation is accomplished by

$$A \text{Mod}(2^n + 1) = A \text{Mod} 2^n - A \text{div} 2^n \quad (5)$$

where div represents division retaining only the quotient.

3.0 An Algorithm for Diminished-1 Multiplication

Our algorithm assumes that neither the multiplier or multiplicand is zero; i.e. zero detection has been completed prior to using this algorithm. These are the same assumptions used in many previously reported diminished-1 multipliers, and so our final comparisons will be valid.

To derive the general formulae for multiplication, we start with the special case of multiplication by 2^k .

From eqn. (2), we find:

$$d(2A) = d(A + A) = 2d(A) + 1 \pmod{2^n + 1} \quad (6)$$

Since $2^k A$ can be treated as adding A to itself $2^k - 1$ times, then:

$$d(2^k A) = d(A + A + \dots + A) = 2^k d(A) + 2^k - 1 \pmod{2^n + 1} \quad (7)$$

and eqn. (7) can be rearranged as:

$$2^k d(A) = d(2^k A) - 2^k + 1 \pmod{2^n + 1} \quad (8)$$

Representing the n bits of the diminished-1 form as $d(A) = \{a_{n-1}a_{n-2}\dots a_0\}$, then,

from eqn. (7), we have:

$$d(2^k A) = \{a_{n-(k+1)}a_{n-(k+2)}\dots a_0 0 \dots 0\} + \{1 \dots 1\} - \{a_{n-1}a_{n-2}\dots a_{n-k}\} \quad (9)$$

where we use k least significant zeros in the first number and k ones in the second number on the right hand side of eqn. (9). The first number is the binary representation of

$2^k d(A) \text{ Mod}(2^n)$, and the third number is $2^k d(A) \text{ div}(2^n)$. The combination of the last two numbers yields $\{\overline{a_{n-1}a_{n-2}a_{n-3}}\}$, the 1's complement of $\{a_{n-1}a_{n-2}a_{n-3}\}$.

Therefore:

$$d(2^k A) = \{a_{n-4}a_{n-5}\dots a_0 \overline{a_{n-1}a_{n-2}a_{n-3}}\} \quad (10)$$

or in other words, multiplication by 2^k is accomplished by a cyclic shift of k bits to the left with the shifted bits being complemented. This rule was previously derived in [3] by induction. The above argument provides a direct proof of this rule.

Similar to the derivation of eqn. (7), the general multiplication of A , a non-zero number, by B , a non-zero number, can be treated as adding A to itself $B-1$ times. Therefore,

$$d(BA) = Bd(A) + B - 1 = (d(B) + 1)d(A) + d(B) \text{ Mod}(2^n + 1) \quad (11)$$

Representing $d(B)$ by n bits as:

$$d(B) = \left\{ \sum_{k=0}^{n-1} b_k 2^k \right\} \quad (12)$$

we have:

$$\begin{aligned} d(BA) &= \left[\sum_{k=0}^{n-1} b_k 2^k + 1 \right] d(A) + d(B) = \\ &= \sum_{k=1}^{n-1} b_k 2^k d(A) + (b_0 + 1)d(A) + \sum_{k=0}^{n-1} b_k 2^k \text{ Mod}(2^n + 1) \end{aligned} \quad (13)$$

Using eqn. (8), eqn. (13) can be re-written as:

$$d(BA) = \sum_{k=1}^{n-1} b_k (d(2^k A) - 2^k + 1) + (b_0 + 1)d(A) + \sum_{k=0}^{n-1} b_k 2^k \text{ Mod}(2^n + 1) \quad (14)$$

or

$$d(BA) = \sum_{k=1}^{n-1} b_k d(2^k A) + \sum_{k=1}^{n-1} b_k + (b_0 + 1)d(A) + b_0 \text{Mod}(2^n + 1) \quad (15)$$

Let

$$d_1(A) = (b_0 + 1)d(A) + b_0 \text{Mod}(2^n + 1) \quad (16)$$

Then $d_1(A) = d(A)$ if $b_0 = 0$; or $d_1(A) = d(2A)$ if $b_0 = 1$. Or, in other words,

$$d_1(A) = \bar{b}_0 d(A) + b_0 d(2A) \quad (17)$$

Replacing the first summation of eqn. (15) by the diminished-1 summation of eqn. (4), we obtain:

$$d(BA) = \sum_{k=1}^{n-1} \oplus b_k d(2^k A) - n + 2 + \sum_{k=1}^{n-1} b_k + d_1(A) \text{Mod}(2^n + 1) \quad (18)$$

Let Z be the number of zeros of the $n-1$ bits from b_1 to b_{n-1} , then:

$$Z = \sum_{k=1}^{n-1} \bar{b}_k = n - 1 - \sum_{k=1}^{n-1} b_k \quad (19)$$

and

$$d(BA) = \sum_{k=1}^{n-1} \oplus b_k d(2^k A) - Z + d_1(A) + 1 \text{Mod}(2^n + 1) \quad (20)$$

Since $-Z = 2^n + 1 - Z = 2^n - 1 - Z + 2 = \bar{Z} + 2 \text{Mod}(2^n + 1)$, then:

$$d(BA) = \sum_{k=1}^{n-1} \oplus b_k d(2^k A) + \bar{Z} + d_1(A) + 3 \text{Mod}(2^n + 1) \quad (21)$$

or

$$d(BA) = \sum_{k=1}^{n-1} \oplus b_k d(2^k A) \oplus \bar{Z} \oplus d_1(A) + 1 \text{ Mod}(2^n + 1) \quad (22)$$

Except for the final binary addition of 1, which can be easily implemented by setting the carry-in of the final diminished-1 adder, all operations in eqn. (22) are diminished-1 operations. If $b_k=1$, we implement $d(2^k A)$ by cyclic shifts together with a complement operation on the shifted bits, as shown previously. If $b_k=0$, we replace $d(2^k A)$ with n zeros. Therefore, the $n+1$ numbers (excluding the last 1) on the right hand side of eqn. (22) are all n -bit diminished-1 numbers.

4.0 A New Architecture for Modulo (2^n+1) Multiplication

The efficient implementation of multi-operand diminished-1 addition is of great importance to the speed of the modulo $(2^n + 1)$ multiplier. In the published literature, the addition of diminished-1 numbers is always implemented by self-contained diminished-1 adders, which require an n -bit binary addition plus a carry correction. Since a carry correction is equivalent to an n -bit half adder, which has a critical path delay similar to an n -bit binary full adder, the self-contained n -bit diminished-1 adder is considerably slower than an n -bit binary adder. If we look at the multiplication as a multi-operand adder tree, however, then we can invoke transformations which effectively remove the extra delay associated with the diminished-1 adder structure. To start with let us use an example to demonstrate the existing techniques of modulo $(2^n + 1)$ multiplication.

4.1 The existing techniques

Let us perform multi-operand addition on three modulo (2^8+1) diminished-1 numbers 108(01101011), 47(00101110), and 158(10011101).

First, we use the original diminished-1 addition scheme given in [3]. As shown by Fig. 1, this scheme requires two n -bit binary additions and two carry corrections.

$$\begin{array}{r}
 01101011 \quad 108 \\
 + 00101110 \quad 47 \\
 \hline
 010011001 \\
 + \quad \quad \quad \rightarrow \\
 \hline
 10011010 \quad 155 \\
 + 10011101 \quad 158 \\
 \hline
 100110111 \\
 + \quad \quad \quad \rightarrow 0 \\
 \hline
 00110111 \quad 56 \text{ mod } 257
 \end{array}$$

Figure 1. Adding three diminished-1 numbers using the scheme given in [3]

In general, if K diminished-1 numbers are added with this scheme, $K-1$ n -bit binary additions and the same number of carry corrections are required. Let t_a be the delay of an n -bit addition and t_c be the delay of a carry correction, then the delay of this scheme is given by:

$$T_1 = (K - 1)(t_a + t_c) \quad (23)$$

The scheme proposed in [7] is shown in Fig. 2, where the number of carry corrections has been reduced from 2 to 1. In fact, no matter how many numbers are added, only one carry correction is required, since all other carry corrections can be merged with the binary additions. Therefore, if K diminished-1 numbers are added by this scheme, $K-2$ carry corrections can be saved in comparison with the first scheme, but $K-1$ n -bit binary additions are still to be done. The delay of this scheme is given by:

$$T_2 = (K - 1)t_a + t_c \quad (24)$$

$$\begin{array}{r}
 01101011 \quad 108 \\
 + 00101110 \quad 47 \\
 \hline
 010011001 \quad 155 \\
 | 10011101 \quad 158 \\
 + \quad \quad \quad \rightarrow 1 \\
 \hline
 100110111 \\
 + \quad \quad \quad \rightarrow 0 \\
 \hline
 00110111 \quad 56 \text{ mod } 257
 \end{array}$$

Figure 2. Adding three diminished-1 numbers using the scheme given in [7]

4.2 Application of the Wallace Tree

It is well known that multi-operand binary addition can be optimally implemented by a Wallace tree [12]. The optimality is achieved at the expense of an irregular interconnection structure compared to slower regular array solutions [13]. To take the advantage of the Wallace tree structure in our design, the only extra problem we have to take care of is the carry correction. That is, to complement any carry out of the MSB in our n -bit representation, and shift it to the LSB position.

Fig. 3 uses the new scheme proposed in this paper in the example of Section 4.1.

Each bit of the three numbers is added by a binary adder. The sum bit of the adder is fed to the same bit position of an n -bit fast adder, while the carry bit of the adder is fed to the n -bit adder by shifting one bit position to the left for all adders, except the adder in the MSB position; in this latter case the carry is complemented and fed to the LSB position of the n -bit fast adder. This fast adder then adds the sums and carries together with a carry correction to yield the final result.

$$\begin{array}{r}
 01101011 \quad (108) \\
 00101110 \quad (47) \\
 + 10011101 \quad (158) \\
 \hline
 S \quad 11011000 \\
 C \quad 00101111 \\
 \xrightarrow{\hspace{1.5cm}} \\
 10011011 \\
 \xrightarrow{\hspace{1.5cm}} 0 \\
 \hline
 00110111 \quad (56)
 \end{array}$$

Figure 3. Adding three diminished-1 numbers using the new scheme

In general, if K diminished-1 numbers are to be added, we treat the same bit position of the set of input numbers as a column. Then we use a Wallace tree [12] of binary adders, to compress the column size from K to 2. The number of stages, j , required by the Wallace tree is determined by the recursion of eqn. (25):

$$\begin{aligned}
 K(0) &= 2 \\
 K(j+1) &= \left\lfloor \frac{3}{2} K(j) \right\rfloor
 \end{aligned} \tag{25}$$

where $\lfloor \cdot \rfloor$ represents the integral part of the argument (floor function). The series so generated is:

j	1	2	3	4	5	6	7	8	9	...
$K(j)$	3	4	6	9	13	19	28	42	63	...

Let the inverse function of $K(j)$ be $j(K)$. Then $j(K)$ is given by Table 1.

Table 1. Number of stages as a function of column height

K	3	4	5-6	7-9	10-13	14-19	20-28	29-42	43-63	...
$j(K)$	1	2	3	4	5	6	7	8	9	...

Thus, $j(K)$ stages are required by the Wallace tree in order to compress the column size from K to 2. Since $j(K) \sim \log_{1.5} K$, then $j(K) \ll K$ for large K .

When the Wallace tree is applied to the specific problem of diminished-1 multi-operand addition, the carries of adders at the MSB position of any stage have to be complemented and fed to the input of adders at the LSB position of the next stage.

Let t_o be the delay of a binary adder. Then the delay for adding K diminished-1 numbers by our scheme is given by:

$$T_3 = j(K)t_o + t_a + t_c \tag{26}$$

Since $t_o < t_a$, and $j(K) < K$, then $T_3 < T_2$ and our scheme appears to be the fastest proposed so far.

4.3 An example

Figure 4 shows an example of modulo $(2^8 + 1)$ multiplication using the proposed algorithm. The multiplicand and the multiplier are 59 (000111010) and 184 (010110111), respectively. The final result is 62 (000111101) Mod 257. In Figure 4, the generation of the partial product array, i.e. $d_1(A)$, $b_k d(2^k A)$, and \bar{Z} is given by **A**. The Wallace tree, is given by **B**, **C**, **D**, and **E**. The final diminished-1 addition, with a carry-in (the last term in eqn. (22)) is given by **F** (binary addition) and **G** (carry correction). All bits on the left of

the broken lines are complemented and shifted to the right hand side (identified by borders on the top and left). In the Wallace tree stage, every three rows of partial products are compressed to two rows by a full adder array. The resulting two rows are marked by ‘s’ for sums, and ‘c’ for carries, the carry row is shifted one bit position to the left with respect to the sum row.

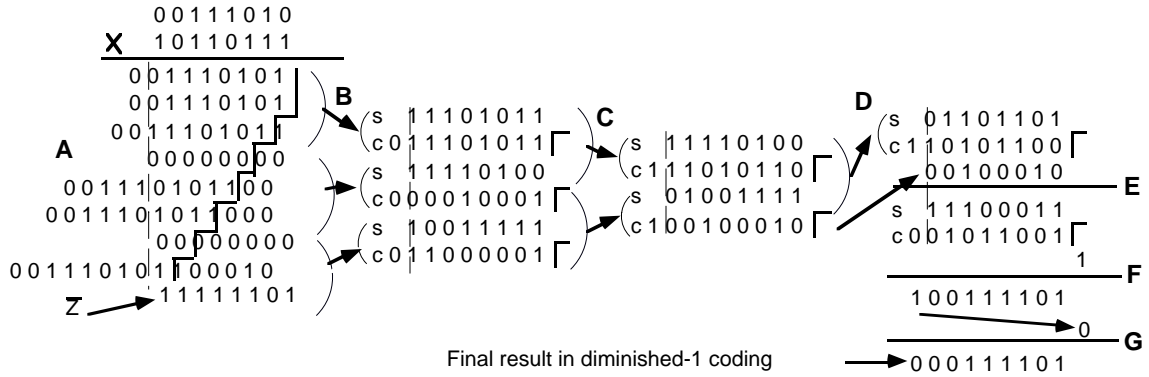


Figure 4. An example of diminished-1 multiplication.

4.4 Computing Z

The number Z is obtained as the result of a counter, which counts the number of zeros of the $N-1$ bits from b_1 to b_{n-1} . Z must be computed before the Wallace tree is applied. For the example of $N-1=7$, the counter can be implemented as shown by Figure 5. Taking advantage of the difference in delay between the carry and sum of a full adder [16], the seven counter has the equivalent delay of four XOR gates.

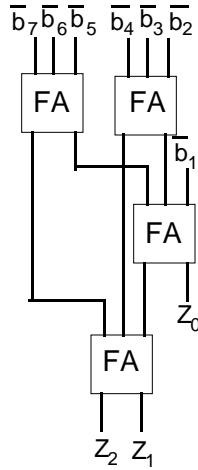


Figure 5. The implementation of a seven counter by full adders

4.5 An architecture for Modulo 257 multiplication

Figure 6 shows the architecture for the new modulo $(2^n + 1)$ multiplier for $n=8$, where each block represents a full adder. The inputs of the adder enter from the top, while the outputs, the sum on the right side and carry on the left, exit from the bottom. The small circle at the carry position of each of the most left adders represents a complemented output. a_k and $b_k, k = 0, 1, \dots, n - 1$ are diminished-1 representation of bits taken from n LSBs of the multiplicand and multiplier, respectively.

It is well known that when the Wallace tree is used to build a binary multiplier, the irregularity of the architecture and interconnection cause layout difficulties. In contrast, it can be seen from Figure 6 that the full adder array for the Wallace tree in our architecture is extremely regular, and is therefore easier to implement than the Wallace tree binary multiplier, especially when n becomes large. This, we feel, is a rather intriguing result.

5.0 Comparison with Published Techniques

For comparison purposes, we use the algorithm proposed in [11] based on a study presented in reference [10], since this appears to represent the fastest published modulo

$(2^n + 1)$ multiplier technique.

this clearly renders our technique superior to prior results, since we remove the conversion overhead.

Since our modulo $(2^n + 1)$ multiplier is based on the Wallace tree we will compare it to an $n \times n$ Wallace tree binary multiplier.

Both our multiplier and the $n \times n$ bit Wallace tree binary multiplier consist of three sections: partial product generation; the Wallace tree, which reduces the column size of the partial products from n to 2; and a fast adder, which combines the carries and sums from the second part into a single number. Our comparison is made for each part separately.

For a binary multiplier, all partial products are obtained by AND gates. The delay is one AND gate delay. The partial products for our modulo $(2^n + 1)$ multiplier, on the other hand, require some extra hardware and longer delay for generation of Z , as has been shown in Section 4.4. Comparing to the whole multiplier, the extra hardware is negligible. Although the extra delay is not negligible (for the example of $N=8$, an extra delay equivalent to 3.5 XOR delays is required), it is, however, still a small portion of the total delay of the multiplier.

The number of full adders for the Wallace tree is $(n - 1) \times (n - 2)$ for the binary multiplier [14], while that for our modulo $(2^n + 1)$ multiplier is $n \times (n - 1)$, (easily obtained from Figure 6). Because of the irregularity of the Wallace tree for a binary multiplier, compared to the regularity of our multiplier, more silicon area will be spent on routing, especially when n becomes large. It is difficult, in general, to show the area trade-off of the extra $2(n - 1)$ full adders required by our multiplier versus the much greater wiring irregularity of the binary multiplier, since these are very technology dependent. It is clear, however, that the trade-off tends to reduce the effect of the extra full adders for our modulo $(2^n + 1)$ multiplier.

The delay of the Wallace tree depends on the height of the column of the partial product array (Table 1), providing that we do not count the delay of the wiring. The column size for an $n \times n$ bit binary multiplier is n , while the column size for the modulo $(2^n + 1)$ multiplier is $n+1$. For many n in practical use (e.g. 8 and 16) $j(n + 1) = j(n)$, yielding

the same delay for the second part of both multipliers. The irregular routing and interconnection of the binary multiplier will, in fact, tend to increase its delay, offsetting the effect of the first stage delay increase for the modulo $(2^n + 1)$ multiplier.

For the third part we compare a $2(n - 1)$ bit binary adder for the binary multiplier, to an n -bit diminished-1 adder for the modulo $(2^n + 1)$ multiplier. The latter consists of two n -bit adders (full and half). For fast adders, such as carry-lookahead or carry-skip, a $2(n - 1)$ bit adder requires more hardware than two n bit adders. The delay of a fast adder is approximately proportional to $\log_2 n$ [15], and so our delay comparison is approximately $\log_2 n + 1$ vs. $2\log_2 n$. Although the delay of an n -bit half adder is shorter than that of an n -bit full adder, clearly the delay of the third part of our modulo $(2^n + 1)$ multiplier is longer than that of the binary multiplier.

Considering that the second part of the multiplier has a major impact on the area and delay, then we have the interesting result that our Wallace tree modulo $(2^n + 1)$ multiplier has similar area and delay to that of a Wallace tree $n \times n$ binary multiplier.

6.0 Conclusions

In conclusion, an architecture for a modulo $(2^n + 1)$ multiplier is presented. The multiplier is based on a modified Wallace Tree, rather than on binary multipliers and converters, as proposed in the literature. An interesting observation is that the resulting Wallace tree is much more regular than Wallace Tree structures applied to binary multipliers. The required hardware and the delay of the new modulo $(2^n + 1)$ multiplier is similar to that required by a $n \times n$ bit Wallace tree binary multiplier. The regularity of the proposed modulo $(2^n + 1)$ multiplier suggests suitability for VLSI implementation. The proposed modulo $(2^n + 1)$ multiplier appears to be superior to published modulo $(2^n + 1)$ multiplier architectures in terms of both hardware and speed.

7.0 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.

8.0 References

- [1] R. C. Agarwal and C. S. Burrus: "Fast convolution using Fermat number transforms with applications to digital filtering", *IEEE Trans. Acoust. Speech, Signal Processing*, vol ASSP-22, pp. 87-97, 1974.
- [2] W. Luo, G.A. Jullien, N.M. Wigley, W.C. Miller, Z. Wang: "An Array Processor for Inner Product Computations Using a Fermat Number ALU" *Proc. 1995 Symposium on Application Specific Array Processors* (in print).
- [3] 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.
- [4] W. K. Jenkins: "The design of specialized residue classes for efficient recursive digital filter realization", *IEEE Trans. Acoust. Speech, Signal Processing*, vol ASSP-30, pp. 370-380, 1982.
- [5] W. K. Jenkins: "Recent advance in residue number techniques for recursive digital filtering", *IEEE Trans. Acoust. Speech, Signal Processing*, vol ASSP-27, pp. 19-30, 1979.
- [6] F. J. Taylor: "A VLSI residue arithmetic multiplier", *IEEE Trans. Comput.*, vol. C-31, pp. 540-546, 1982.
- [7] J. J. Chang, T. K. Truong, H. M. Shao, I. S. Reed, and L. -S. Hsu: "The VLSI design of a single chip for the multiplication of integers modulo a Fermat number", *IEEE Trans. Acoust. Speech, Signal Processing*, vol. ASSP-33, pp. 1599-1602, 1985.
- [8] M. Benaissa, A Pajayakrit, S. S. Dlay, and A. G. J. Holt: "VLSI design for diminished-1 multiplication of integers modulo a Fermat number" *IEE Proc.* vol. 135, Pt. E. pp. 161-164, 1988.

- [9] M. Benaissa, A Bouridane, S. S. Dlay, and A. G. J. Holt: “Diminished-1 multiplier for a fast convolver and correlator using the Fermat number transform” *IEE Proc.*, vol. 135, Pt. G. pp. 187-193, 1988.
- [10] A. V. Curiger, H. Bonnenberg and H. Kaeslin: “Regular VLSI architecture for multiplication modulo (2^n+1) ”, *IEEE J. Solid-State Circuits*, vol. 26, pp. 990-994, 1991
- [11] A. Hiassat: “New memoryless mod (2^n+1) residue multiplier” *Elec. Letts.*, vol. 28, pp. 3124-315, 1992.
- [12] C. S. Wallace: A suggestion for a fast multiplier, *IEEE Trans. Electronic Computers*, vol. EC-13, pp. 14-17, 1964.
- [13] J. Y. Lee, H. L. Garvin and C. W. Slayman: A high-speed high-density silicon 8x8-bit parallel multiplier, *IEEE J. Solid-State Circuits*, vol. SC-22, 35-40, 1987.
- [14] Zhongde Wang, G.A. Jullien, W.C. Miller: New Design Techniques for Column Compression Multipliers, *IEEE Trans. Comput.*, vol. C-44, No. 8, pp. 962-970, 1995
- [15] R. P. Brent and H. T. Kung: A regular layout for parallel adders, *IEEE Trans. Comput.*, vol. C-31, pp. 260-264, 1982
- [16] V. G. Oklobdzija, D. Villeger, and S. S. Liu: “A method for speed optimized partial product reduction and generation of fast parallel multiplier using an algorithmic approach”, *IEEE Trans. Computers.*, vol. C-45, No. 3, pp. 294-306, 1996.

PostScript error (--nostringval--, --nostringval--)