New Multiplier Designs Based on Squared Law Algorithms and Table Look-ups

Poornachandra B. Rao and Alexander Skavantzos

Louisiana State University

Abstract

A new multiplier design based on look-up tables and squared law algorithms is presented. This multiplier can compute the full precision product of two integer numbers or their product modulo $2^N$, modulo $2^N - 1$, or modulo $2^N + 1$. The design is based on decomposing the two numbers into smaller sizes, performing the cyclic convolution on these smaller numbers and then finally reconstructing the desired product. The algorithms use no multiplications and instead rely only on squaring operations, additions, and subtractions. A comparative analysis of ROM requirements and hardware complexities for performing the multiplication operation using various look-up techniques is presented.

1.0: Introduction

The multiplication operation is one of the four basic operations and is used extensively, both in general purpose and special purpose computing. As such there exists a vast amount of literature on the variety of multiplication algorithms, references[1]-[4] to name a few. Recently new multipliers modulo $(2^N - 1)$[5] and modulo $(2^N + 1)$[6] have been developed. Apart from the requirements of computer arithmetic, the fields of digital signal processing and cryptography have several algorithms that perform arithmetic in modular rings[7]-[8]. Multipliers designed using look-up tables[2],[5],[9] offer attractive speed-redundancy trade offs[6], however their main draw back has been excessive ROM sizes and thus the inability to integrate the entire design on a single chip. This problem was addressed in [5] and it was shown how the huge ROM size problem can be circumvented by breaking down the numbers into smaller components, performing cyclic convolution between these smaller numbers and then finally reconstructing the desired product. Reference[5] also describes the importance of using only squaring operations for performing the multiplication operation.

The organization of the paper is as follows. Section 2.0 of the paper provides the necessary theorems and proofs for the computation of cyclic convolutions using principally squaring operations. The heart of the multiplier is a modulo $2^N - 1$ multiplier and the design of this is presented in section 3.0. The design is presented for the case when the numbers are decomposed into four parts. This is done solely for the purpose of clarity and can easily be extended to a general case. Section 4.0 extends the modulo $2^N - 1$ multiplier to obtain a modulo $2^N + 1$ multiplier, Section 5.0 presents a hardware-speed analysis and finally section 6.0 summarizes the paper.

2.0: Mathematical Basis

Consider two sequences each of length n-points given as $A = \{a_0, a_1, ..., a_{n-1}\}$ and $B = \{b_0, b_1, ..., b_{n-1}\}$. Then the cyclic convolution between these two sequences can be given as an n-point sequence $C = \{c_0, c_1, ..., c_{n-1}\}$ with each $c_i$ defined by

$$c_i = \sum_{k=0}^{n-1} a_{i-k} \cdot b_k \quad \text{for } i = 0, 1, ..., n-1 \quad (1)$$

In the above and in the rest of the paper $\langle x \rangle_m$ denotes the operation $x \mod m$ between integers. With reference to the above computation the following theorems apply.

Theorem 1: Assume that $n$ is even and define

$$x_{n1} = a_0 + a_2 + ... + a_{n-2} + b_0 + b_2 + ... + b_{n-2} \quad (2)$$

$$x_{n2} = a_0 + a_2 + ... + a_{n-2} - b_0 - b_2 - ... - b_{n-2} \quad (3)$$

$$x_{n3} = a_1 + a_3 + ... + a_{n-1} + b_1 + b_3 + ... + b_{n-1} \quad (4)$$

$$x_{n4} = a_1 + a_3 + ... + a_{n-1} - b_1 - b_3 - ... - b_{n-1} \quad (5)$$

Then

$$\sum_{i=0}^{n-1} x_{ni}^2 = x_{n2}^2 + x_{n3}^2 + x_{n4}^2 + \sum_{i=0}^{n-1} = 4 \sum_{i=0}^{n-1} c_{2i}$$

$$= 4(c_0 + c_2 + ... + c_{n-2}) \quad (6)$$
Proof:

\[ x_{n1}^2 - x_{n2}^2 + x_{n3}^2 - x_{n4}^2 = (x_{n1} + x_{n2})(x_{n1} - x_{n2}) + (x_{n3} + x_{n4})(x_{n3} - x_{n4}) \]
\[ = \sum_{i=0}^{n/2-1} b_{2i} + \sum_{i=0}^{n/2-1} b_{2i+1} \]
\[ = 4 \left( \sum_{i=0}^{n/2-1} a_{\text{even}}^\text{even} \right) + \sum_{i=0}^{n/2-1} a_{\text{odd}} \text{b}_{\text{odd}} \]
\[ = 4 \left( \sum_{i=0}^{n/2-1} c_{2i} \right) \]

and the proof of (6) is completed.

Note that \( c_{2i} = \sum a_v b_w \) such that \( <v+w>_n = 2i \) or \( v+w = nx + 2i \). But since \( n = \) even it implies that \( v+w = \) even and therefore \( v \) and \( w \) are either both even or both odd. This justifies the last step of the proof.

**Theorem 2:** Assume that \( n \) is even and define

\[ y_{n1} = a_0 + a_2 + ... + a_{n-2} + b_1 + b_3 + ... + b_{n-1} \]  
\[ y_{n2} = a_0 + a_2 + ... + a_{n-2} - b_1 + b_3 + ... + b_{n-1} \]  
\[ y_{n3} = a_1 + a_3 + ... + a_{n-1} + b_0 + b_2 + ... + b_{n-2} \]  
\[ y_{n4} = a_1 + a_3 + ... + a_{n-1} - b_0 - b_2 + ... + b_{n-2} \]

Then

\[ y_{n1}^2 - y_{n2}^2 + y_{n3}^2 - y_{n4}^2 = 4 \sum_{i=0}^{n/2-1} c_{2i+1} \]
\[ = 4(c_1 + c_3 + ... + c_{n-1}) \]

Proof: The proof is similar to that of (6) and is thus omitted.

In reference[5] equations (6) and (11) were realized using four squares, each being a function of all \( 2n \) points of the sequences. Here we use more squares but optimize each equation and thereby reduce the total number of ROM bits as shown in detail in section 5.0. Another advantage of using these new theorems is that the computation of the even points of the convolved sequence is kept independent from that of the odd points thus enabling us to further parallelize the hardware architecture.

**Theorem 3:** Assume that \( n > 2, n = 2^p \Rightarrow n = 4m \), and define

\[ z_{n1} = a_0 - a_2 + a_4 - a_6 + ... - a_{n-2} + b_0 - b_2 + b_4 - b_6 + ... - b_{n-2} \]  
\[ z_{n2} = a_0 - a_2 + a_4 - a_6 + ... - a_{n-2} - b_0 + b_2 - b_4 + b_6 - ... + b_{n-2} \]  
\[ z_{n3} = a_1 - a_3 + a_5 - a_7 + ... + a_{n-1} - b_1 + b_3 + b_5 + b_7 - ... + b_{n-1} \]  
\[ z_{n4} = a_1 - a_3 + a_5 - a_7 + ... + a_{n-1} + b_1 + b_3 + b_5 + b_7 - ... + b_{n-1} \]

Then

\[ z_{n1}^2 - z_{n2}^2 + z_{n3}^2 + z_{n4}^2 = 4(c_0 - c_2 + c_4 - c_6 + ... - c_{n-2}) \]

Proof: The proof can be constructed on lines similar to that of (6) and is thus omitted.

**Theorem 4:** Assume that \( n > 2, n = 2^p \Rightarrow n = 4m \), and define

\[ z_{n5} = a_0 - a_2 + a_4 - a_6 + ... - a_{n-2} + b_1 - b_3 + b_5 - b_7 + ... - b_{n-1} \]  
\[ z_{n6} = a_0 - a_2 + a_4 - a_6 + ... - a_{n-2} - b_1 + b_3 - b_5 + b_7 - ... + b_{n-1} \]  
\[ z_{n7} = a_1 - a_3 + a_5 - a_7 + ... - a_{n-1} + b_0 - b_2 + b_4 - b_6 + ... - b_{n-2} \]  
\[ z_{n8} = a_1 - a_3 + a_5 - a_7 + ... - a_{n-1} - b_0 + b_2 - b_4 + b_6 + ... + b_{n-2} \]

Then

\[ z_{n5}^2 - z_{n6}^2 + z_{n7}^2 - z_{n8}^2 = 4(c_1 - c_3 + c_5 - c_7 + ... - c_{n-1}) \]

Proof: The proof can be constructed on lines similar to that of (6) and is thus omitted.

3.0: Modulo \( 2^N - 1 \) Multiplication

Consider the multiplication of two \( N \)-bit binary numbers \( A \) and \( B \). Let each of the numbers be decomposed into four parts given as \( [a_3, a_2, a_1, a_0] \) and \( [b_3, b_2, b_1, b_0] \). The number \( A \) is then given as \( a_32^{3N/4} + a_22^{2N/4} + a_12^{N/4} + a_0 \), and number \( B \) can be evaluated in a
similar fashion. Then their product modulo $2^N -1$ can be given as
\[ \langle A \times B \rangle_{2^N -1} = \langle c_0 + c_1 2^{N/4} + c_2 2^{N/2} + c_3 2^{3N/4} \rangle_{2^N -1} \] (22)
with $c_0$, $c_1$, $c_2$, and $c_3$ defined as
\[ c_0 = a_0 b_0 + a_3 b_1 + a_2 b_2 + a_1 b_3 \] (23)
\[ c_1 = a_1 b_0 + a_0 b_1 + a_3 b_2 + a_2 b_3 \] (24)
\[ c_2 = a_2 b_0 + a_1 b_1 + a_0 b_2 + a_3 b_3 \] (25)
\[ c_3 = a_3 b_0 + a_2 b_1 + a_1 b_2 + a_0 b_3 \] (26)

Note that equations (23)-(26) can also be generated by (1). Thus we can apply theorems (1)-(4) to obtain $c_0$, $c_1$, $c_2$, and $c_3$. Define
\[ x_{41} = a_0 + a_2 + b_0 + b_2 \] (27)
\[ x_{42} = a_0 + a_2 - b_0 - b_2 \] (28)
\[ x_{43} = a_1 + a_3 + b_1 + b_3 \] (29)
\[ x_{44} = a_1 + a_3 - b_1 - b_3 \] (30)
\[ y_{41} = a_0 + a_2 + b_1 + b_3 \] (31)
\[ y_{42} = a_0 + a_2 - b_1 - b_3 \] (32)
\[ y_{43} = a_1 + a_3 + b_0 + b_2 \] (33)
\[ y_{44} = a_1 + a_3 - b_0 - b_2 \] (34)
\[ z_{41} = a_0 - a_2 + b_0 - b_2 \] (35)
\[ z_{42} = a_0 - a_2 - b_0 + b_2 \] (36)
\[ z_{43} = a_1 - a_3 + b_1 + b_3 \] (37)
\[ z_{44} = a_1 - a_3 - b_1 - b_3 \] (38)
\[ z_{45} = a_0 - a_2 + b_1 - b_3 \] (39)
\[ z_{46} = a_0 - a_2 - b_1 + b_3 \] (40)
\[ z_{47} = a_1 - a_3 + b_0 - b_2 \] (41)
\[ z_{48} = a_1 - a_3 - b_0 + b_2 \] (42)

Then theorems (1)-(4) give
\[ x_{41}^2 - x_{42}^2 + x_{43}^2 - x_{44}^2 = 4(c_0 + c_2) \] (43)
\[ y_{41}^2 - y_{42}^2 + y_{43}^2 - y_{44}^2 = 4(c_1 + c_3) \] (44)
\[ z_{41}^2 - z_{42}^2 - z_{43}^2 + z_{44}^2 = 4(c_0 - c_2) \] (45)
\[ z_{45}^2 - z_{46}^2 + z_{47}^2 - z_{48}^2 = 4(c_1 - c_3) \] (46)

Thus we get
\[ c_0 = \frac{1}{8}(x_{41}^2 - x_{42}^2 + x_{43}^2 - x_{44}^2 + z_{41}^2 - z_{42}^2 + z_{43}^2 + z_{44}^2) \] (47)
\[ c_1 = \frac{1}{8}(y_{41}^2 - y_{42}^2 + y_{43}^2 - y_{44}^2 + z_{45}^2 - z_{46}^2 + z_{47}^2 - z_{48}^2) \] (48)

\[ c_2 = \frac{1}{8}(x_{41}^2 - x_{42}^2 + x_{43}^2 - x_{44}^2 - z_{41}^2 + z_{42}^2 + z_{43}^2 - z_{44}^2) \] (49)
\[ c_3 = \frac{1}{8}(y_{41}^2 - y_{42}^2 + y_{43}^2 - y_{44}^2 - z_{45}^2 + z_{46}^2 - z_{47}^2 + z_{48}^2) \] (50)

Each square required by equations (47)-(50) is realized using a ROM. The advantages of such techniques are detailed in references [2]-[5]. Although the number of squares required is more than that of [5] the total number of ROM bits required is less than that required by [5]. Hardware in terms of adders and subtracters is comparable with that required by [5]. Section 5.0 offers a detailed comparative analysis.

4.0: Extending the modulo $2^N -1$ multiplier

Continuing with the same notation as before, the modulo $2^N +1$ product of two numbers A and B can be given as
\[ \langle A \times B \rangle_{2^N +1} = \langle d_0 + d_1 2^{N/4} + d_2 2^{N/2} + d_3 2^{3N/4} \rangle_{2^N +1} \] (51)
with $d_0$, $d_1$, $d_2$, and $d_3$ defined as
\[ d_0 = a_0 b_0 - a_3 b_1 - a_2 b_2 - a_1 b_3 \] (52)
\[ d_1 = a_1 b_0 + a_0 b_1 - a_3 b_2 - a_2 b_3 \] (53)
\[ d_2 = a_2 b_0 + a_1 b_1 + a_0 b_2 + a_3 b_3 \] (54)
\[ d_3 = a_3 b_0 + a_2 b_1 + a_1 b_2 + a_0 b_3 \] (55)

Note that term $d_3$ of equation (55) is the same as term $c_3$ of equation (26) and so no extra ROM bits are required for computing $d_3$. To compute $d_0$, $d_1$, and $d_2$ define
\[ g_0 = a_3 b_1 + a_2 b_2 + a_1 b_3 \] (56)
\[ g_1 = a_3 b_2 + a_2 b_3 \] (57)
\[ g_2 = a_3 b_3 \] (58)

Then
\[ d_0 = g_0 \cdot 2g_0 \] (59)
\[ d_1 = g_1 \cdot 2g_1 \] (60)
\[ d_2 = g_2 \cdot 2g_2 \] (61)

The terms $g_0$, $g_1$, and $g_2$ can be computed by directly applying the quarter squared algorithm[2]-[4] while these equations are not presented here. Doubling these terms can be obtained by simply shifting the numbers to
the left by one position and thus this needs no extra ROM bits. In a similar fashion the full precision and modulo $2^N$ product of two numbers can be computed. It will be seen that these computations will require no additional ROM bits.

5.0: Hardware and Speed Analysis

All the analysis in this section is provided for the case when the product of two numbers is obtained by decomposing each number into four equal parts, say each with $k$ bits. Four methods are compared:

i) traditional techniques,
ii) quarter squared algorithm,
iii) new multipliers modulo $2^N -1$ [5],
iv) techniques of this paper.

Table 1 summarizes the results and also presents data on the number of adders required by each method. Here we have assumed that the operation $(a - b)$ requires only one adder. Method (i) requires a ROM whose size is in the order of $O(2^{2k})$ while all the other three methods require ROMs with sizes of order $O(2^k)$. Since method (i) requires the largest sized ROMs, it will be the slowest. Also since the total number of ROM bits is very high, it will not be possible to integrate the entire design on a single chip. With respect to speed and number of ROM bits, methods (ii) through (iv) are comparable. With respect to the total number of ROM bits required, method (ii), i.e. the direct application of the quarter squared algorithm, appears to be the best but considering the fact that it requires 60 adders it will be the most complex one to build. Our techniques in this paper out perform those of [5] in many respects viz. smaller maximum ROM size, a total of fewer ROM bits, and higher speed. Also based on the techniques of this paper, all the ROMs are of the same size and hence identical. This again is a big advantage for VLSI designs.

Theorem 5: The cyclic convolution of two four point sequences, each point with $k$ bits, can be computed with no less than 40 two-operand adders and $(k + 2) x 2^{k+7}$ ROM bits.

Proof: Equations (47)-(50) show that the required computation can be evaluated by computing 16 squares as given by equations (27)-(42). In the evaluation of equations (47)-(50) not a single product term of the form $aibj$ is cancelled, i.e. none of the ROM bits are used for unnecessary computations. Hence proved.

Table 2 summarizes the ROM requirements for different word lengths ($N$) for the case when the numbers are each decomposed into four parts. Table 3 summarizes the overhead ROM requirements required for computing the product modulo $2^N +1$, modulo $2^N$, and the full precision product. Overhead is defined as the number of ROM bits needed over and above those required for the computation of the modulo $2^N -1$ product. The values are based on equations (56)-(58), are obtained in a straightforward manner and are hence not detailed.

6.0: Summary

In this paper we have provided fast and efficient hardware multiplier designs based on cyclic convolutions, squaring operations, and table look-ups. We presented detailed designs for modulo $2^N -1$ and modulo $2^N +1$ multipliers. The designs were based on decomposing the numbers into four parts. Several theorems have been provided, which can be used to extend the decomposition to a general case. We have also compared several look-up table techniques and have shown that our techniques result in ROM bit savings of 99.99% when compared with traditional techniques.

Acknowledgement

This work was supported by the National Science Foundation and Louisiana Stimulus for Excellence in Research under Grants CCR8910187 and NSF/LaSER (1991)-RCD-01.

REFERENCES

Table 1: Hardware and speed comparison of various look-up table techniques for computing the product of two numbers modulo \(2^N - 1\).

<table>
<thead>
<tr>
<th>Method</th>
<th>Total number of ROM bits</th>
<th>Number of two operand adders</th>
<th>Size of largest ROM</th>
<th>Speed as a fn. of ROM size</th>
<th>Integration on a single chip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trad. Techniques</td>
<td>(k \times 2^{2k+5})</td>
<td>12</td>
<td>(2^{2k} \times 2^k)</td>
<td>(O(2^{2k}))</td>
<td>NO, for (N \geq 16)</td>
</tr>
<tr>
<td>Quarter Sq. Alg.</td>
<td>((k+1) \times 2^{k+7})</td>
<td>60</td>
<td>(2^{k+1} \times (2k+2))</td>
<td>(O(2^{k}))</td>
<td>NO, for (N \geq 16)</td>
</tr>
<tr>
<td>Reference[5]</td>
<td>((2k+5) \times 2^{k+6})</td>
<td>38</td>
<td>(2^{k+3} \times (2k+6))</td>
<td>(O(2^{k}))</td>
<td>Possible</td>
</tr>
<tr>
<td>Tech. of this paper</td>
<td>((k+2) \times 2^{k+7})</td>
<td>40</td>
<td>(2^{k+2} \times (2k+4))</td>
<td>(O(2^{k}))</td>
<td>Possible</td>
</tr>
</tbody>
</table>

Table 2: Cost Comparison in ROM bits of the various techniques for computing \(\langle A \times B \rangle \cdot 2^N - 1\) with each of the numbers decomposed into four equal parts.

<table>
<thead>
<tr>
<th>Word Length N</th>
<th>Decomposed Part Length k</th>
<th>Traditional Techniques cost</th>
<th>Quarter Squared Alg. cost</th>
<th>Reference[5] cost</th>
<th>Techniques of this paper cost</th>
<th>% savings (this paper vs. trad. tech.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>4</td>
<td>2^5 \times 2^{10}</td>
<td>10 \times 2^{10}</td>
<td>13 \times 2^{10}</td>
<td>12 \times 2^{10}</td>
<td>62.50</td>
</tr>
<tr>
<td>32</td>
<td>8</td>
<td>2^{10} \times 2^{14}</td>
<td>18 \times 2^{14}</td>
<td>21 \times 2^{14}</td>
<td>20 \times 2^{14}</td>
<td>98.04</td>
</tr>
<tr>
<td>64</td>
<td>16</td>
<td>2^{19} \times 2^{22}</td>
<td>34 \times 2^{22}</td>
<td>37 \times 2^{22}</td>
<td>36 \times 2^{22}</td>
<td>99.99</td>
</tr>
</tbody>
</table>

Table 3: Cost in ROM bits for integrated multiplier, based on techniques of this paper. Each product is computed by decomposing each of the numbers into four equal parts.

<table>
<thead>
<tr>
<th>Word Length N</th>
<th>Decomposed Part Length k</th>
<th>Mod. ((2^N - 1)) product cost ((k+2) \times 2^{k+7})</th>
<th>Mod. ((2^N + 1)) product overhead cost (6(k+1) \times 2^{k+3})</th>
<th>Mod. ((2^N)) product overhead cost</th>
<th>Full precision product overhead cost</th>
<th>% overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>4</td>
<td>6 \times 2^{11}</td>
<td>30 \times 2^7</td>
<td>None</td>
<td>None</td>
<td>31.25</td>
</tr>
<tr>
<td>32</td>
<td>8</td>
<td>10 \times 2^{15}</td>
<td>54 \times 2^{11}</td>
<td>None</td>
<td>None</td>
<td>33.75</td>
</tr>
<tr>
<td>64</td>
<td>16</td>
<td>18 \times 2^{23}</td>
<td>102 \times 2^{19}</td>
<td>None</td>
<td>None</td>
<td>35.41</td>
</tr>
</tbody>
</table>