Implementation of Proposed Interleaved Karatsuba Montgomery Multiplier- A Review

Lawhale Tejashree#1, Mrs. Kasliwal Prabha#2,

#1M.E. Student, Electronics Department, MIT AOE, Pune University, Maharashtra, India
#2Associate Prof., Electronics Department, MIT AOE, Pune University, Maharashtra, India

Abstract— Modular multiplication is a core operation in public-key cryptosystems. We propose to implement the Interleaved Karatsuba Montgomery Multiplier (IKM) combined with Karatsuba-Ofman and Montgomery algorithm to accelerate modular multiplication on FPGA. The Karatsuba-Ofman algorithm is fast multiplication algorithm with the complexity of \(O(\log^3 N/\log 2)\). The performance parameter in terms of delay, area and power dissipation will be compared with published results of Montgomery algorithm.

Index Terms—Public key Cryptography, Modular Multiplication, Karatsuba-Ofman algorithm, Montgomery Modular multiplication.

I. INTRODUCTION

Cryptography is the practice and study of techniques for secure communication in the presence of third parties called adversaries. The rising growth of data communication and electronic transactions over the internet has made security to become the most important issue over the network. To provide modern security features, public-key cryptosystems are used. The widely used algorithms for public-key cryptosystems are RSA, Diffie-Hellman key agreement (DH), the Digital signature algorithm (DSA) and systems based on Elliptic curve cryptography (ECC). Long word lengths are necessary to provide a sufficient amount of security, but it also account for the computational cost of these algorithms. The cryptography systems practically depend on how fast the modular multiplication is done and consequently a high performance long integer multiplication is in demand.

Modular Multiplication [1] \(A \times B \mod M\) can be performed in two different ways: multiplying, i.e. computing \(P = A \times B\); and reducing, i.e. \(R = P \mod M\) or interleave the multiplication and the reduction steps.

The prominent algorithms are Karatsuba-Ofman’s method for multiplying and Montgomery algorithms methods for interleaving multiplication and reduction.

The most popular algorithm to perform modular arithmetic to date is Montgomery Modular Multiplication (MMM). MMM involves three multiplications, therefore the acceleration of the multiplication can also benefit the performance. The multiplication algorithms faster than the Schoolbook multiplication are listed in Table I.

Karatsuba multiplication is asymptotically an \(O(\log^3 N/\log 2)\) algorithm, the exponent being \(\log (3)/\log (2)\), representing 3 multiplies each ½ the size of the inputs. This is a big improvement over the base case multiply at \(O(N^2)\) and the advantage soon overcomes the extra additions Karatsuba performs [3].

In this paper, we analyze the characteristics of the Karatsuba-Ofman algorithm, and apply its idea in MMM to perform modular multiplications by interleaving the computation. The steps to implement the proposed algorithm:

- Interleaved Karatsuba Montgomery Multiplier algorithm (IKM) will be proposed for efficient modular multiplication computation.
- VHDL implementation of IKM algorithm.
- Results will be simulated.
- The results will be compared with the published Montgomery Algorithms.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schoolbook</td>
<td>(O(N^2))</td>
</tr>
<tr>
<td>Karatsuba</td>
<td>(O(N^{\log 3/\log 2}))</td>
</tr>
<tr>
<td>Toom-Cook</td>
<td>(O(N^{\log (2k-1)/\log k}))</td>
</tr>
<tr>
<td>Schonhage-Strassen</td>
<td>(O(N \log N \log \log N))</td>
</tr>
<tr>
<td>Furer</td>
<td>(O(N \log N \log \log N))</td>
</tr>
</tbody>
</table>

Fig. 1. Block diagram of Modular Multiplication

Table I
MULTIPLICATION ALGORITHMS AND THEIR COMPLEXITY [2]
II. MONTGOMERY MODULAR MULTIPLICATION

One of the widely used algorithms for efficient modular multiplication is the Montgomery Modular Multiplication algorithm.

Algorithm 1: Standard Montgomery Modular Multiplication [4]

<table>
<thead>
<tr>
<th>Inputs: X, Y, M with 0 &lt; X, Y &lt; M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: P = (X * Y * (2^n) - 1) mod M</td>
</tr>
<tr>
<td>n: number of bits in X;</td>
</tr>
<tr>
<td>x_i: i bit of X;</td>
</tr>
<tr>
<td>p_0: LSB of P;</td>
</tr>
<tr>
<td>(1) P: = 0;</td>
</tr>
<tr>
<td>(2) for (i = 0; i &lt; n; i + +) {</td>
</tr>
<tr>
<td>(3) P: = P + x_i * Y;</td>
</tr>
<tr>
<td>(4) P: = P + p_0 * M;</td>
</tr>
<tr>
<td>(5) P: = P div 2; }</td>
</tr>
<tr>
<td>(6) if (P ≥ M) then P: = P - M;</td>
</tr>
</tbody>
</table>

The Montgomery algorithm, [Algorithm 1] computes P = (X * Y * (2^n) - 1) mod M. The idea of Montgomery is to keep the lengths of the intermediate results smaller than n + 1 bit. This is achieved by interleaving the computations and additions of new partial products with divisions by 2; each of them reduces the bit length of the intermediate result by one.

The Montgomery algorithm is very easy to implement since it operates least significant bit first and does not require any comparisons. A modification of Algorithm 1 with carry save adders is given in Algorithm 2. The core operation of most algorithms for modular multiplication is addition. There are several different methods for addition in hardware: carry ripple addition, carry select addition, and carry look ahead addition and others.

The disadvantage of these methods is the carry propagation, which is directly proportional to the length of the operands. This is not a big problem for operands of size 32 bits but the typical operand size in cryptographic applications range from 160 to 2048 bits. The resulting delay has a significant influence on the time complexity of these adders. The carry save adder seems to be the most useful adder for modular multiplication architectures [5]. Figure 2 gives us the flowchart of Montgomery Modular Multiplication.


<table>
<thead>
<tr>
<th>Inputs: X, Y, M with 0 &lt; X, Y &lt; M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: P = (X * Y * (2^n) - 1) mod M</td>
</tr>
<tr>
<td>n: number of bits in X;</td>
</tr>
<tr>
<td>x_i: i bit of X;</td>
</tr>
<tr>
<td>s_0: LSB of S, c_0: LSB of C, y_0: LSB of Y;</td>
</tr>
<tr>
<td>R: precomputed value of Y + M;</td>
</tr>
<tr>
<td>(1) S:= 0; C:= 0;</td>
</tr>
<tr>
<td>(2) for (i = 0; i &lt; n; i + +) {</td>
</tr>
<tr>
<td>(3) if (!s_0 &amp;&amp; c_0) and not x_i then I:= 0;</td>
</tr>
<tr>
<td>(4) if (!s_0 &amp;&amp; c_0) and not x_i then I:= M;</td>
</tr>
<tr>
<td>(5) if (not (s_0</td>
</tr>
<tr>
<td>(6) if (s_0</td>
</tr>
<tr>
<td>(7) S, C = S + C + I;</td>
</tr>
<tr>
<td>(8) S:= S div 2; C:= C div 2;</td>
</tr>
<tr>
<td>(9) P:= S + C;</td>
</tr>
<tr>
<td>(10) if (P ≥ M) then P:= P - M;</td>
</tr>
</tbody>
</table>

So to improve the runtime Karatsuba multiplier is used with the combination of Montgomery Modular Multiplication.

III. KARATSUBA-OFSMAN ALGORITHM

Karatsuba-Ofman’s algorithm is one of the fastest ways to multiply long integers. It is even faster than Schonhage-Strassen’s method and is based on a divide-and-conquer strategy [6]. A multiplication of a 2n-digit integer is reduced to one (n+1)-digit multiplication, two left-shift operations, two n-digits additions two n-digits subtractions, and two 2n-digits.

The details of Karatsuba-Ofman’s multiplication algorithm are as follows,

Let X and Y be the binary representation of two integers:
The Karatsuba-Ofman algorithm uses divide and conquer method. The operands \( X \) and \( Y \) can be decomposed into \( X_H, X_L, Y_H \) and \( Y_L \) respectively.

\[
X = \sum_{i=0}^{k-1} x_i 2^i \quad \text{and} \quad Y = \sum_{i=0}^{k-1} y_i 2^i
\]

The product \( P = XY \) can be computed as follows:

\[
P = (X_H Y_H + X_L Y_L + X_H Y_L + X_L Y_H)
\]

Using the above equation, it needs four \( n \) bits multiplications to compute the product \( P \). The standard multiplication is based on that equation. So assuming that a multiplication of \( k \)-bits operands is performed using \( T(k) \) one bit operations. We can also write \( T(k) = T(n) + \delta k \), where \( \delta k \) is a number of one bit operations to compute all the additions and shift operations. When \( T(1) = 1 \), we find that the standard multiplication algorithm requires,

\[
T(k) = (k \log_2 4)^2 = (k^2)
\]

The computation of \( P \) can be enhanced by,

\[
X_H Y_H + X_L Y_L = (X_H + X_L)(Y_H + Y_L) - (X_H Y_H) - (X_L Y_L)
\]

On the observations made above the Karatsuba-Ofman’s algorithm is based, so the \( 2n \) bits multiplication can be reduced to \( 3n \) bits multiplication which are \( (X_H + X_L)(Y_H + Y_L), (X_H Y_H) \) and \( (X_L Y_L) \). Algorithm 3 gives us the Karatsuba-Ofman multiplication method.

**Algorithm 3** KaratsubaOfman \((X, Y)\) \([2]\)

If \((\text{Size}(X)=1)\) Then KaratsubaOfman=OneBitMultiplier(X, Y)
Else Product1:= KaratsubaOfman(High(X), High(Y));
Product2:= KaratsubaOfman(Low(X), High(Y));
Product3:=KaratsubaOfman(High(X)+Low(X),High(Y)+
Low(Y));
KaratsubaOfman:= Rightshift(Product1, Size(X)) +
Rightshift(Product3-Product1-Product2,Size(X/2))
+Product2;1
End KaratsubaOfman.

The Karatsuba-Ofman computes the Algorithm 3. The signals \( SY_L \) and \( SX_L \) in figure 3 are two \( n \)-bits result of the additions \((Y_H + Y_L)\) and \((X_H + X_L)\) respectively. The CY and CX represent the one-bit carry out of these additions. The ShiftAdd first computes \( S \) sum as \( SX_L + SY_L, SX_L, SY_L \) or 0 depending on the values of \( CY \) and \( CX \). The Product3 is then computed and \( T \) represents \( CX \times CY \). The ShiftSubAdd first computes \( R = Product_1 + Product_2 \), then obtain \( 2CR \) which is \( 2 \)’s complement of \( R \). This gives us \( Product_3 = Product_3 + 2CR \). After shifting \( Product_3 \) and \( U \) left to \( 2n \) times and \( n \) times respectively, the components reduces the first and last additions as well as the shift operations in the last line computation of Algorithm 3.

The product of two large numbers is the basic step of Karatsuba’s algorithm. The \( \text{Size} \) \((X)\) returns the number of bits of \( X \), function \( \text{High}(X) \) returns the higher half part of \( X \). Rightshift(X, n) returns \( X \) rightn, function \( \text{Low}(X) \) returns the lower half of \( X \). OneBitMultiplication(X,Y) return \( X \) when both \( X \) and \( Y \) are formed by single bit. If \( \text{Size}(X) \) is odd, then \( \text{High}(X) \) and \( \text{Low}(X) \) right-pad \( X \) with a zero before extracting the high and the low half respectively. The algorithm above requires \( 3n \) \(-\)bits multiplications to compute \( P \). So we can write that,

\[
T(k) = 2T(n) + T(n+1) + \delta k \approx 3T(n) + \delta k
\]

Where \( \delta k \) is the number of one bit operations to compute all the additions, shift operations and subtractions. When \( T(1)=1 \) the algorithm requires,

\[
T(k) = (k \log_2 3) = (k^{1.58})
\]

So it is asymptotically faster than the standard multiplication algorithm.
IV. INTERLEAVED KARATSUBA MONTGOMERY MULTIPLIER

The Karatsuba-Ofman algorithm was developed by Karatsuba and Ofman in 1962 and performs large integer multiplication in fewer operations. It is the key to our Montgomery Modular Multiplication algorithm. To get a high performance, we will implement Interleaved Karatsuba Montgomery Multiplication algorithm which combines both Karatsuba-Ofman algorithm and Montgomery Modular Multiplication.

Algorithm 4 Interleaved Karatsuba Montgomery Multiplier

<table>
<thead>
<tr>
<th>INPUT: R= b^n, integer m with n radix b digits and gcd(m,b) = 1, positive integer x and y with n radix b digits and x&lt;m.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT: x<em>y</em>R⁻¹ mod m.</td>
</tr>
</tbody>
</table>

Karatsuba(x,y)
for (i from 0 up to s-1)
C:= 0
m:= t[i] *n[i] mod W
for (j from 0 up to s-1)
(C,S) := t[i+j] + m*n[j] + C
t[i+j] := S
end for
ADD(t[i+j],C)
end for
for (j from 0 up to s)
u[j] := t[j+s]
end for
B:= 0
for (i from 0 up to s-1)
(B,D) := u[i] – n[i] – B
end for
if B=0 return t[0],t[1],……..t[s-1]
else return u[0],u[1],……..u[s-1]

Algorithm 4 gives us the IKM Algorithm. Modular Multiplication [7] can be performed in two different ways: multiplying, then reducing, or interleave the multiplication and the reduction steps. For multiplying we use Karatsuba-Ofman multiplication algorithm, and use the Montgomery algorithms methods for interleaving multiplication and reduction.

A modular reduction [8, 9] is the computation of the remainder of an integer division. It is given by,

\[ X \mod M = X - \left\lfloor \frac{X}{M} \right\rfloor \times M \]

Division is very expensive even compared with a multiplication. The sequential division algorithm successively shifts and subtracts the modulus is found. A negative remainder might be obtained after subtraction, for this case the last non negative remainder should be restored and that will be the expected remainder. This computation is described in the above algorithm. This computation requires n comparisons, n subtractions and some restoring operations also some 2n shifting. This is hence very much more efficient than the previous algorithm.

V. IMPLEMENTATION AND EXPECTED RESULTS

We will implement Interleaved Karatsuba Montgomery Multiplier Algorithm on a Xilinx SPARTAN-3/Virtex-6 device. We will compare the results with previously existing algorithms. Virtex-6 device has 18-bit signed multiplier. We will use it to develop the Karatsuba pipelined multiplier. To increase the frequency the base multipliers in DSP48E1 is pipelined with three stages.

We use VHDL to describe the IKM architecture and the functional simulation is done using Modelsim Simulator. The test bench for top level design, along with stimulus will be described. Simulation results and timing diagrams will be compared with the previous algorithms. It will show that our algorithm is faster than the previous algorithms and also hardware friendly.

CONCLUSION

We will propose that the performance evaluation is efficient using Interleaved Karatsuba Montgomery Multiplier algorithm in this paper. The parameter requirements of the proposed algorithm will be analyzed. The experimental results of previous Montgomery modular multipliers will be compared with our IKM designs. We will refine the architecture design, and speedup the accumulation process.
ACKNOWLEDGMENT

I would like to acknowledge and extend my heartfelt gratitude to my guide Prof. Mrs. Prabha Kasliwal and our H.O.D of Electronics Dept, Dr. M. D. Gaudor. I am also thankful to our M.E. Coordinator Prof. Satyajit Pangaonkar for his valuable information and timely cooperation. I am also very thankful to our Principal Dr. Y. J. Bhalerao for providing us with an environment to complete my project successfully.

REFERENCES


