Architectures for fast modular multiplication

Ahmet Aris and Berna Ors
Faculty of Electrical and Electronics Engineering
Istanbul Technical University
Istanbul, Turkey
Email: arisahmet@itu.edu.tr

Gokay Saldamli
Department of MIS
Bogazici University
Istanbul, Turkey
gokay.saldamli@boun.edu.tr

Abstract—Modular multiplication is the key ingredient needed to realize most public-key cryptographic primitives. In a modular setting, multiplications are carried out in two steps: namely, a usual integer arithmetic followed by a reduction step. Progress in any of these steps naturally improves the modular multiplication but it is not possible to interleave the best algorithms of these stages. In this study, we propose architectures for recently proposed method of interleaving the Karatsuba-Ofman multiplier and bipartite modular reduction on the uppermost layer of Karatsuba-Ofman’s recursion. We manage to come up with a high performance modular multiplication architecture by taking advantage of a fast multiplication and a parallel reduction method.

Keywords—Bipartite modular multiplication, Karatsuba-Ofman Algorithm, modular arithmetic, interleaved multiplication.

I. INTRODUCTION

Applied sciences have various practices which require sophisticated applications of sophisticated mathematics. For instance, in cryptography, most public-key algorithms including RSA [1] and ECC [2] involve advanced arithmetic operations with large moduli. Modular multiplication is the main focus of related research in cryptographic engineering as it is the basic operation in exponentiation and inversion calculations.

In modular world, multiplication and reduction stages are interleaved in order to reach compact and scalable hardware designs. However, asymptotically fast multipliers such as Karatsuba-Ofman (KO) [3] or FTT based methods (e.g. Schönhage-Strassen [4] and Fürer [5]) could not be interleaved with any reduction methods. Here the main problem is the dependency issue of the reduction algorithms that does not allow parallel processing (i.e. parallel reduction). However, a recent study [6] demonstrates a partial method of interleaving a fast multiplier with a modular reduction. To be specific: the author combines the well known KO multiplier with the bipartite reduction on the uppermost layer of KO’s recursion. Bipartite Modular Multiplication (BMM) method (see [7], [8]) which is originally introduced by Kaihara and Takagi, is based on the observation that, a product can simultaneously be reduced from left and right without a dependency issue. Although dependency exits within each direction, BMM algorithm outlines a global method of parallel reduction.

In this study we propose architectures for the partially interleaved KO multiplier [6]. Taking the advantage of a fast multiplication and parallel BMM reduction, we manage to reduce the number of clock cycles and got a faster modular multiplier. In the remaining of this presentation, we will give brief descriptions of BMM and related preliminaries in Section 2. Proposed method that merges KO multiplier with the bipartite reduction in the uppermost level of KO’s recursion will be outlined in Section 3. In Section 4, hardware implementation and evaluations based on this method will be proposed.

II. MODULAR ARITHMETIC

Modular arithmetic finds various applications and studies in the field of applied sciences. In the literature, there are various proposals and enhancements for carrying modular multiplication. A simple classification of these methods is done with respect to their reduction approach; namely, algorithms reducing from left-to-right and from right-to-left.

For left-to-right approach, several proposals ([9], [10], [11], etc.) exist. Algorithms including the standard division can be put into this category.

Montgomery multiplication [12] is the only example of right-to-left reduction. It computes the modular product of two integers without performing divisions by the modulus N. In fact, it replaces the costly divisions with trivial divisions by a power of 2. Technical report [13] includes a very clear presentation of the method. For our needs, we give Algorithm 2.1 (i.e. the radix-2 version of Montgomery’s multiplication algorithm [14]), which calculates the Montgomery product of two positive integers [15].

Algorithm 2.1: Montgomery Modular Multiplication.

Require: \( N = (n_{k-1} \ldots n_1 n_0)_2, \ A = (a_{k-1} \ldots a_1 a_0)_2, \ B = (b_{k-1} \ldots b_1 b_0)_2, \ r = 2^k \mod N, \ n_0 = 1 \)

Ensure: \( R = \text{Montgomery}(A, B, N) = A.B.r^{-1} \mod N \)

for \( i = 0 \) to \( k - 1 \) do
\( q_i = (r_0 + a_i \times b_0) \mod 2 \)
\( R = (R + a_i \times B + q_i \times N) / 2 \)
end for
III. BIPARTITE KARATSUBA-OFTMAN MULTIPLIER

A. Bipartite Modular Multiplication

BMM method introduced by Kaihara and Takagi in [7] and [8], presents a semi parallel reduction based on an observation that a product could simultaneously be reduced from left and right without a dependency issue. Although, the dependency exists within each direction, BMM algorithm outlines a global method of parallel reduction.

\[ \begin{align*}
B_0 := 2^h & \quad \text{and} \quad B_1 := 2^h B_1 + B_0, \\
A := 2^h I_1 + I_0 & \quad \text{and} \quad A := 2^h I_1 + I_0.
\end{align*} \]

Figure 1. Bipartite reduction combining left-to-right and right-to-left reductions.

Since left-to-right and right-to-left reductions do not have any dependency during the first half of their reduction steps, they could be combined as seen in Fig. 1. In fact, the figure gives a sketch of the bipartite reduction.

The strength of BMM is clear; since the reduction is split into two parts, it can be handled separately in parallel. Therefore, theoretically, BMM should shrink the reduction time by half. In [8], the authors report the performance figures of BMM realizations.

B. Karatsuba-Ofman Multiplication

KO algorithm presents a recursive method which requires asymptotically fewer bit operations than the standard multiplication. For a brief explanation, firstly, we decompose \( a \) and \( b \) into two equal-size parts: \( A := 2^h A_1 + A_0 \) and \( B := 2^h B_1 + B_0 \), i.e., \( A_1 \) and \( A_0 \) represent the respective most and least significant \( h \) bits of \( a \), assuming \( k \) is even and \( 2^h = k \).

The KO algorithm is based on the following observation that, in fact, three half-size multiplications suffice to compute the product \( A \times B \).

\[ \begin{align*}
T_0 := A_0 \cdot B_0 \\
T_2 := A_1 \cdot B_1 \\
T_1 := (A_0 + A_1)(B_0 + B_1) - T_0 - T_2 \\
&= A_0 \cdot B_1 + A_1 \cdot B_0.
\end{align*} \]

In depth details of the algorithm can be found in their original paper [3] and in Knuth [16]. Note that, one has the option of stopping at any point during the recursion. For example, we may apply one level of recursion and then compute the required three multiplications using the standard nonrecursive multiplication algorithm.

C. Partially interleaved KO multiplication

In this section, we outline the method called partially interleaved KO multiplication presented in [6]. As mentioned earlier, general reduction routine is not lucky in terms of data dependency, as a result, it can not be parallelized and interleaved with fast multipliers. However, [6] interleaves bipartite reduction with KO multiplier to an extend. To be specific, BMM can be interleaved with KO on the upper most layer of KO recursion.

Let \( Q, M \) and \( Q' \) be as follow:

\[ \begin{align*}
N := 2^h N_1 + N_0 &= r N_1 + N_0, \\
Q := 2^h Q_1 + Q_0 &= r Q_1 + Q_0, \\
Q' := 2^h Q'_1 + Q'_0 &= r Q'_1 + Q'_0.
\end{align*} \]

If the following partial products are defined:

\[ \begin{align*}
T_0' := A_0 \cdot B_0 - Q_0' \cdot N_0 &= T_0 - Q_0 \cdot N_0, \\
T_2' := A_1 \cdot B_1 - Q_1' \cdot N_1 &= T_2 - Q_1 \cdot N_1.
\end{align*} \]

Using these values, \( T_1' \) can be calculated as follows:

\[ \begin{align*}
T_1' := T_1 + T_0' + T_2' - Q_0' N_1 - Q_1 N_0 &\equiv (A_0 + A_1)(B_0 + B_1) - T_0 - T_2 \\
&\quad + T_0' + T_2' - Q_0' N_1 - Q_1 N_0 \\
&\quad + (A_0 + A_1)(B_0 + B_1) - T_0' + T_2' - Q_0' N_0 - Q_1 N_1 \\
&\quad + T_0' - T_0 + T_2' - T_2' - T_2'.
\end{align*} \]

Observe that, only 6 half-size multiplications are needed for \( T_1' = ABr^{-1} \mod N \) (i.e the modular multiplication) calculation. As seen in the block diagram given in Fig. 2, the method can be realized in a highly parallel fashion. The top block which consists of two paired multiplications (in case, able to work in parallel) implements the idea of BMM but unlike BMM these multiplications are half-sized. Meanwhile, the block underneath implements two integer multiplications that can run in parallel. Moreover, since they are not modular, further KO recursion could be employed.

IV. HARDWARE IMPLEMENTATION

As it can be seen in Fig. 2, hardware implementation of partially interleaved KO multiplier consists of one half-sized classical(Blakley) modular multiplier, one half-sized Montgomery multiplier, seven registers where four of them are half sized registers storing the values coming from Montgomery and classic modular multiplier modules in the first part of the design. In the second part, there are four adders, in addition to two integer multipliers operating in parallel and performing integer multiplication. In the last part, a modular adder and a register for intermediate results are accommodated for final modular additions.
In our half-sized Montgomery and classic multiplier modules, calculation of quotient values \( Q_1 \) and \( Q'_0 \) is an essential process which makes our architecture somewhat different than very well known Montgomery and classic modular multiplier architectures. These quotient values are used in integer multipliers later on and give us the chance of \( T'_1 \) reduction which is result of modular multiplication \( ABr^{-1} \mod N \). Details with implementation of these multiplier blocks are given in the following sections.

A. Classic Modular Multiplier Module

Our classical multiplier employs the estimation logic proposed in the work of Bunimov et al. [17]. This estimation logic uses the most significant two bits of intermediate result and modulus. By using this approach, it is possible to reduce the subtractions to a single subtraction. Moreover, instead of two full-bits-lengthed comparisons with modulus, only a comparison to \( k \cdot 2^n (k = 0, 1, \cdots, 6) \) is performed at each step, which can be done in constant time, as values of \( k \cdot 2^n \mod M \) are precomputed before the execution of the loop as shown in Fig. 3.

B. Montgomery Multiplier Module

Block diagram of Montgomery multiplier is shown in Fig. 4. At each step, a partial product \( I \), some multiple of modulus \( M \) and \( A \) are added using the CSA which is composed of 5-to-3 compressors, and sum(S) and carry(C) are generated. Value of \( A \) is determined by the least significant bits of partial product and sum at each step. Here number of modulo \( M \)'s that are added during the multiplication steps and subtracted at the end of the algorithm determine the value of quotient \( Q'_0 \).

V. Tests and Results

We described our design in VHDL and implemented on Xilinx Virtex 5 XCVFX130T FPGA. Verification of correct operation of the design was done according to the modular multiplication values from the software implementation of the design in Maple.

For the \( n \)-bit operands \( X, Y \) and modulo \( N \), our Montgomery multiplier implementation performs one modular multiplication operation in \((35n/64) + 18 \) clock cycles. Classic modular multiplier performs modular multiplication in \((79n/128) + 49 \) clock cycles. Integer multiplier module performs multiplication in \((65n/256) + 6 \) clock cycles. As Montgomery multiplier and Classic modular multiplier work in parallel, first part of the design performs its operation in \((79n/128) + 49 \) clock cycles. Two Integer Multiplier modules work in parallel in the second part of the design and complete the integer multiplication in \((65n/256) + 6 \) clock cycles. Based on these values, our multiplier architecture performs modular multiplication in \((263n/256) + 98 \) clock cycles.

Total number of clock cycles, minimum period, maximum frequency, total computation time and area results of 512, and 1024 bit implementation are shown in the Table I.
VI. CONCLUSION

In this work, we described architectures for a modular multiplication method, which combines the ideas of the recently proposed partially interleaved Karatsuba-Ofman multiplier. The method presents an interleaved processing on the upper most level of KO’s recursion.

Note that all the presented work is based on the realizations on Xilinx FPGA families. As a future work, we are planning to work with ASIC platforms in order to enjoy further optimizations for faster timing performance.

ACKNOWLEDGMENT

The authors would like to thank to the anonymous reviewers for their helpful comments in improving this presented material. Note that, the authors are partially funded by TUBITAK research project No: 109E180.

REFERENCES


