MONTGOMERY MODULAR MULTIPLIER ARCHITECTURES AND HARDWARE IMPLEMENTATIONS FOR AN RSA CRYPTOSYSTEM

A. P. Fournaris and O. Koufopavlou
Electrical and Computer Engineering Department
University of Patras, Patras, GREECE
apofour@ee.upatras.gr

Abstract - *This paper describes and analyses the Montgomery Multiplication Algorithm and proposes two scalable, systolic architectures and hardware implementations based on this algorithm in order to be used for an RSA module. The Conventional Architecture uses the original version of Montgomery Multiplication Algorithm and the Optimized Architecture a modified version of the algorithm. The second architecture is considerably better than the first one Both architectures follow Carry – Save redundant logic and in comparison with other known architecture give interesting results in term of clock frequency, Multiplication time and chip Covered area..*

I. INTRODUCTION

One of the most popular public key cryptographic algorithms is RSA. It is extensively used in public signatures applications and generally in communication transactions. It offers good cryptographic security although because of the demanding mathematical calculation complexity it lacks in speed in comparison with symmetric key algorithms. Thus the need for speeding up the calculations for RSA cryptosystem is great.

The mathematics behind that algorithm is summarized in two operations, modular multiplication and modular exponentiation. In the RSA cryptosystem, the arithmetic operation $A^C \text{mod} N$ is used, where N is a prime product of two relative prime numbers, A is the message and C the secret key. As can be seen in order to create an efficient implementation of RSA one has to design efficiently the multiplication of two modular numbers. However modular multiplication has a very big drawback, trial division has to be employed to obtain the necessary remainder value.

Division is a very consuming operation in terms of time and complexity. The classical school algorithm [1] is practically unusable for modern applications and alternative methods have been devised in order to come with the quotient and remainder of the operation.

The solution to the modular multiplication problem is to find a method of bypassing the trial division operation. Many algorithms have been designed following that logic [1]. However the most popular and most useful one, is the Montgomery Multiplication Algorithm [2]. The algorithm normalizes the result of the multiplication and reduces the complexity of Modular Multiplication especially if large numbers are involved like in public key cryptography - RSA.

The main advantage of the Montgomery Multiplication against other modular multiplication algorithms is that with simple mathematical operations (like addition and multiplication) and little processing cost, it can give very accurate results. The need for division or even inversion is degenerated in a shift action.

Many implementations of the algorithm have been proposed [3], for hardware or software. Some implementations use high radix representations in expense of extra complexity while others rearrange or modify the algorithm so as to improve the performance.

In this paper two scalable systolic Montgomery Modular Multiplication Architectures and hardware implementations are proposed with special concern in high speed. The first, conventional, architecture is based on the original algorithm while the second, optimized, architecture is based on an modified version of the Montgomery Multiplication Algorithm which can achieve better results in terms of covered area and speed. Both architectures use redundant Carry-Save arithmetic

The paper is organized as follows. In Section 2 the Montgomery Multiplication Algorithm is analyzed and the modified optimized algorithm is introduced. In Section 3 the two proposed conventional and modified Montgomery Multiplication architectures and hardware implementations are presented. A performance analysis on the proposed architectures and implementations is done in Section 4 and some conclusions are presented in Section 5.

II. MONTGOMERY MULTIPLICATION ALGORITHM

In this Section the Montgomery Multiplication Algorithm is presented and an analysis of improvements on the algorithm is done.

A. The original algorithm

The Montgomery Multiplication Algorithm [2] calculates the value $A = X \cdot Y \cdot R^{-1} \bmod N$ where R is a constant number usually $R=2^n$. The n-bit value N has to be an integer filling the condition gcd(R,N)=1. If N is considered odd, like in RSA, the above constrain is always true. Also because the

algorithm is used for exponentiation the output A is never greater than N so the final subtraction stated in [2] is omitted [4]. So the algorithm becomes

Function MM (X, Y, N)
1. A=0
2. **For** k=0 to n-1 **do begin**
3. $q=(a_0 + x_k y_0) \bmod 2$
4. $A=A+x_k Y+qN$
5. A=A/2
   End
6. **Return** A

Introducing Carry Save redundant logic, the algorithm can be transformed to

Function MMcs (X, Y, N)
1. $C_{2in}=0$, $C_{1in}=0$, $S_{in}=0$
2. **For** k=0 to n-1 **do begin**
3. $q=(S_{in0} + C_{1in0}+C_{2in0} +x_k y_0) \bmod 2$
4. $C_1+C_2+S= C_{2in}+C_{1in}+S_{in}+x_k Y+qN$
5. $C_{2in}=(C_2)/2$, $C_{1in}=(C_1)/2$, $S_{in}=(S)/2$
   End
6. **Return** $C_{2in}/2$, $C_{1in}/2$ and $S_{in}/2$

B. The optimized algorithm.

Observing the original Montgomery Multiplication Algorithm, the function MM consists basically of a loop of additions of A with the values Y, N, Y+N or zero (0). The choice of what value will be added to A depends on the values of x_0 and q. So by knowing x_0 and q one can add only the needed value to A.

Using that observation and the Carry - Save redundant logic, a modified version of the Montgomery Multiplication algorithm can be extracted similar to the MMcs algorithm presented in the previous section.

Function MMCh (X, Y, N)
1. $C_{in}=0$, $S_{in}=0$
2. **For** k=0 to n-1 **do begin**
3. $q=(S_{in0} + C_{in0} +x_k y_0) \bmod 2$
4. **if (x_0 =0) then**
   a. **if (q=0) then**
   b. I=0
   c. **Else**
   d. I=N
   e. **End**
5. **End.**
6. **if (x_0 =1) then**
   a. **if (q=0) then**
   b. I=Y
   c. **Else**
   d. I=Y+N
   e. **End**
7. **End.**
8. $C+S= C_{in}+S_{in}+I$
9. $C_{in}=(C)/2$, $S_{in}=(S)/2$
   End
10. **Return** $C_{in}/2$ and $S_{in}/2$

As can be seen from the above modification the needed additions in one loop cycle of the algorithm are greatly reduced in comparison with the MMcs algorithm. That is done in the expense of an if – state argument. The value Y+N can be precomputed so it does not change efficiency of the algorithm. Overall, however, the gain in additions overcomes the if – state problem and MMCh can be considered, as seen from the following architectures, an optimization on the MM algorithm.

III. ARCHITECTURES AND HARDWARE IMPLEMENTATIONS

In this Section two systolic, scalable architectures are proposed. The conventional Architecture based on the MMcs algorithm of section 2 and the optimized architecture based on MMCh algorithm of Section 2.

A. The conventional architecture.

The MMcs algorithm uses Carry – Save arithmetic to reduce the critical path of the outcome. However as it can be seen in step 4 of the algorithm an addition of 5 numbers is needed to come with a result of 3 values. The additions of Step 3 they are modulo 2 therefore can be degenerated in simple XOR operations. That, however, cannot be done in Step 4 of the algorithm where an addition of 5 numbers is needed to come with a result of 3 values.

The proposed architecture is a systolic one. Every Processing Element (PE) has to perform 5 to 3 addition using the current x and q value. However, the Processing Element of bit 0 has an additional role to perform, the calculation of q value. Therefore the PE of bit 0 has two more gates. The two PE types are shown in Figure 1.

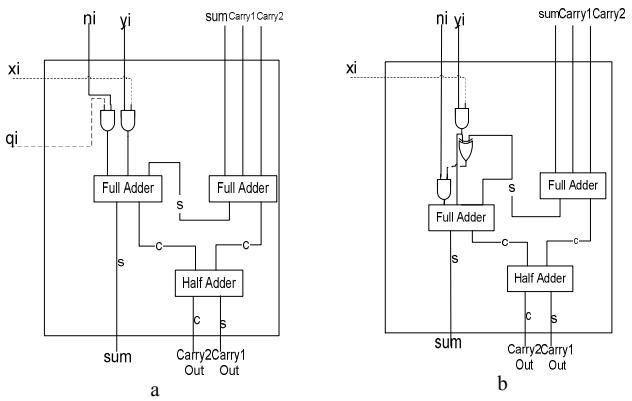

Figure 1.
The Processing Element (a) and the PE for the calculation of q (b) of the Conventional Architecture.

In the systolic architecture presented in Figure 2 the connections of the PE elements have the following logic. Every different column represents the value of the same bit through the algorithmic calculations while each different row represents the values on another round of the MMcs algorithm. The Carry2 signal of a PE is connected with the next PE of the next row of PE, Carry1 signal is connected with the same PE of the next row of PE (the same column) while the Sum signal is connected with the previous PE of the next row starting from bit 0. Using this formula, we manage to avoid the shift operation of Step 5. The outcome is in Carry – Save format through the signals Carry2, Carry1, and Sum.

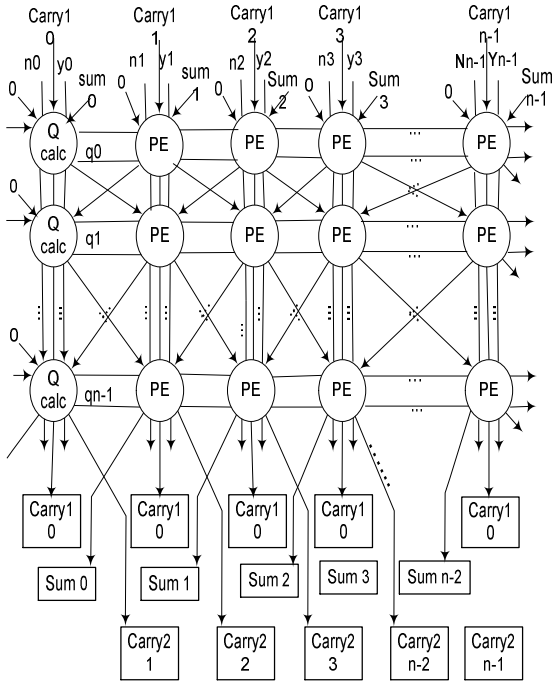

Figure 2
The Systolic Conventional Architecture using algorithm MMcs.

B. The optimized architecture

As seen from algorithm MMCh the number of addition in every PE is less than those in the MMcd algorithm. That remark can lead as to an architecture with much simpler logic. We still however need 2 kinds of Processing Elements; the simple PE and the PE that calculates the q value. The difference is that the PE of the Optimized Architecture has only one Full Adder and a 4-to-1 Multiplexer, as shown in Figure 3,which is a lot more simple than the 2 Full Adder and 1 Half Adder Processing Element of the conventional architecture presented earlier.

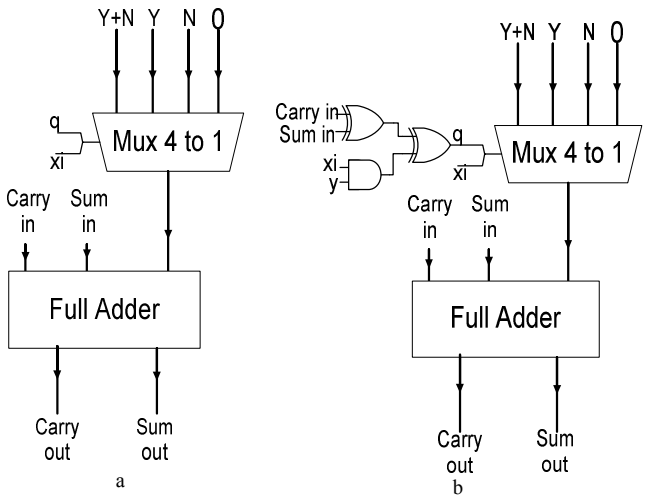

Figure 3
The Processing Element (a) and the PE for the calculation of q (b) of the Optimized Architecture.

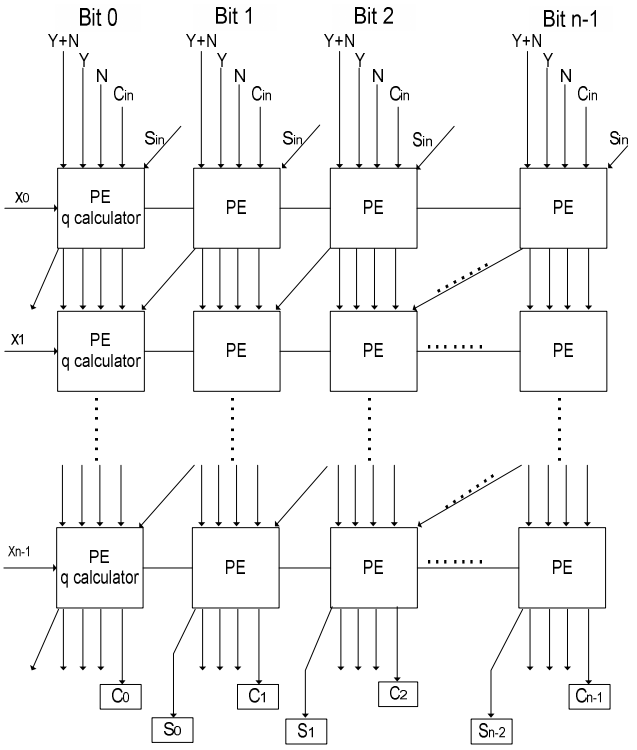

Figure 4
The systolic Optimized Architecture using algorithm MMCh.

The proposed optimized systolic architecture has similar structure as the Conventional Architecture. However the interaction of the Carry – Save signals is much simpler.

Starting from bit 0, the Carry signal is connected with the same PE of the next row of PE (the same column) while the Sum signal is connected with the previous PE of the next row. By that way the shift operation is performed through the correct signal interconnection. The systolic structure of the architecture is shown in Figure 4.

IV. PERFORMANCE AND COMPARISONS

The two proposed architectures where captured in VHDL and implemented in FPGA. A comparison between them was made in terms of frequency and chip covered Area, shown in Table I.

Table I.
FPGA results for a 128 bit implementation of the proposed Architectures.

reference	Conventional	Optimized	Reduction Percentage
Function Generators	130946	65408	50%
CLB's	65473	48767	25.5%
Flip Flops	97534	97534	0%
Clock Frequency	156.3 MHz	168.7MHz	7.35%

The above Table's results prove that the modified version MMCh of the Montgomery Multiplication Algorithm is an optimization of the original MMcs and MM algorithm. It is clearly seen that in Covered Area the MMCh algorithm uses 50% less Function Generators and about 25% less CLB's and in clock Frequency the MMCh algorithm is 7.35% faster. That is a considerable improvement when taking in to account that the proposed multiplier can be a part of an RSA exponentiation module. In such a case small Area and high clock frequency are very crucial factors for the efficiency of the module.

Table II
COMPARISON OF 128 BIT MONTGOMERY MULTIPLIERS

reference	Multiplication Time	Clock Frequency (MHz)
Optimized Architecture	5.12 nsec	168.7
Ors et al. [5]	3.974 μsec	97.63
Nedjah [6]	23 nsec	43.4
Daly [7]	11.2 nsec	88.74

Also in comparison with other Montgomery Multiplier Architectures, as presented in Table II, the optimized Architecture gives formidable results in clock Frequency and Multiplication time. The optimized architecture is faster than all the other compared architectures shown in Table II. However those results come in expense of a high chip covered area value.

V. CONCLUSIONS

The Montgomery Multiplication Algorithm is a very popular solution for the arithmetic operation of Modular Multiplication which is very useful in the RSA public Key Cryptosystem. We proposed two scalable systolic Architectures based on the original and a modified version of the algorithm. Using Carry – Save redundant representation the Montgomery Multiplication Algorithm was accordingly transformed and analyzed. Using the results from the above analysis a Conventional and an Optimized Architecture was proposed and was implemented in FPGA. Comparing the two HW implementations of the Architectures in terms of Clock speed and Chip Covered Area we proved that the Optimized Architecture was better in every way and therefore a good optimization of the original-conventional algorithm. By Comparing the Optimized Architecture with other relative Montgomery Multipliers it is seen that is very advantageous.

VI. REFERENCES

[1] A. J. Menezes, I. F. Blake, X. Gao, R. C. Mullin, S. A. Vanstone, and T. Yaghoobian, *Applications of Finite Fields*, Kluwer Academic, 1993.

[2] Peter L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519-521, 1985.

[3] C. T. Koc, T. Acar and B.S. Kaliski, "Analysing and Comparing Montgomery Multiplication Algorithms", *IEEE micro*, vol. 16, no. 3, pp. 26-33, June 1996.

[4] C. D. Walter, "*Montgomery Exponentiation Needs No Final Subtractions"*, Electronics Letters, vol. **35** no. 21, October 1999, pp 1831-1832.

[5] Siddika Berna Ors, Lejla Batina, Bart Preneel and Joos Vandewalle, "Hardware Implementation of a Montgomery Modular Multiplier in a Systolic Array" *in proc. of International Parallel and Distributed Processing symposium* (IPDPS'03), 2003.

[6] Nadia Nedjah and Luiza de Macedo Mourelle, "Two Hardware Implementations for the Montgomery Modular Multiplication: Sequential versus Parallel", *in proc. of 15th Symposium on Integrated Circuits and System Design* (SBCCI'02), 2002.

[7] Alan Daly and William Marnane, "Efficient Architectures for implementing Montgomery Modular Multiplication and RSA Modular Exponentiation on Reconfigurable Logic", *in proc. of 10th International symposium on Field-programmable gate arrays*, February 2002.