

# Hardware Implementation of Improved Montgomery's Modular Multiplication Algorithm

ZHANG Jia-hong  
Micro-Electricity Center,  
NO.709 Research and  
Development Institute CSIC  
Hubei Wuhan 430074, China  
Zhangjh\_ecc@163.com

Xiong Ting-gang  
Micro-Electricity Center,  
NO.709 Research and  
Development Institute CSIC  
Hubei Wuhan 430074, China

FANG Xiang-yan  
Micro-Electricity Center,  
NO.709 Research and  
Development Institute CSIC  
Hubei Wuhan 430074, China

## Abstract

*This paper describes a hardware implementation of modular multiplication coprocessor for both RSA and ECC Cryptosystems. Using a self-improvement Montgomery modular multiplication algorithm, the coprocessor completes a modular multiplication with less clock cycles under the equivalent circumstance of the other designs. This modular multiplier can deal with variable operand lengths, from 128 to 2048. When adopting 64 bits multiplier, it can work at the frequency of 100MHz targeted to Virtex II XC2V250, and executes 256 bits EC point multiplication, with throughput 172k bit/s and 1024 bits RSA decryption(using CRT), with throughput 483k bit/s.*

## 1. Introduction

RSA<sup>[1]</sup> and ECC<sup>[2][3]</sup> (Elliptic Curve Cryptosystem) are the two major standards used as public-key cryptography<sup>[4]</sup>. In 1978, Rivest, Shamir and Adelman introduced RSA, which is based on the difficult problem of integer factorization. Since these years many breakthrough has taken place in mathematics, the integer length of difficult problem of factorization is increasing (up to now, people can factorize 640 bits number at certain cost). So the key length which is RSA security ground on was required to 1024, even 2048 bits. ECC which has shorter key size was proposed by Koblitz<sup>[2]</sup> and Miller<sup>[3]</sup> individually. As a more secure Public Key Cryptosystems than RSA, ECC was accepted by more and more experts and application domains.

Both of RSA and ECC require fast modular multiplication on precision of 192 to 2048 bits numbers. The modular multiplication algorithm is

always in the important area which people studies. Among those multitudinous algorithms, there are four types of modular multiplication algorithm which are well researched, that are classical algorithm<sup>[5]</sup>, Barret algorithm<sup>[6]</sup>, Montgomery algorithm<sup>[7]</sup> and ZDN algorithm<sup>[8]</sup>. Furthermore, Montgomery algorithm and ZDN algorithm are appropriated to hardware implementation, and our design employs Montgomery algorithm. In this paper, we present an improved Montgomery modular multiplication architecture with a significant advantage in speed and cost among other implementations.

## 2. Background on ECC/RSA

ECC is performed over one of two underlying Galois fields: prime order fields GF (p), and characteristic two fields GF (2<sup>m</sup>). There is no obvious evidence to prove that ECC based on GF(p) is securer than it based on GF(2<sup>m</sup>), but most cryptographist prefer the former for their belief that ECC on GF(p) may bring more handicap for those decoders, and the vital operation in ECC is the same as in RSA. So this paper will concentrate on the arithmetic in GF (p).

An elliptic curve is list as equation (1) over GF(p):

$$y^2 = x^3 + ax + b \pmod{p} \quad (1)$$

where  $a \in F_p, b \in F_p, 4a^3 + 27b^2 \pmod{p} \neq 0$ . The computation of ECC is point add, point double and point multiplication, and all these are based on the arithmetic of GF (p). So the modular operations on GF(p) are very essential, especially the modular multiplication.

RSA also relies on modular multiplication. The basic course of RSA is modular exponential operation which is composite by serial modular multiplication, and equations (2), (3) show it.

$$C = M^e \bmod N \quad (2)$$

$$M = C^d \bmod N \quad (3)$$

### 3. Montgomery modular multiplication algorithm

#### 3.1. Montgomery algorithm

Modular Multiplication is the most crucial operation in both RSA and ECC. The area and performance of the whole chip is directly correlated with it. In 1985, P. L. Montgomery proposed the Montgomery algorithm, and after several betterments this algorithm has been appropriate to hardware implementation<sup>[9][10]</sup>. Given a n bits positive number M(modular) and two n bits operands X and Y,  $0 \leq X, Y < M$ , the Modular Multiplication is to compute,

$$D = X \times Y \pmod{M}, \text{ where } M = \sum_{i=0}^{e-1} m_i B^i,$$

$$0 < m_{e-1} < B, \quad 0 \leq m_i < B \text{ for } i = 0, 1, \dots, e-2,$$

$$X = \sum_{i=0}^{e-1} x_i B^i, Y = \sum_{i=0}^{e-1} y_i B^i, 0 \leq x_i, y_i < B,$$

for  $i = 0, 1, \dots, e-1$ ,  $B = 2^w$ ,  $n = ew$ ,  $w$  is the word size, 32 or 64 usually. As to RSA, n is up to 1024 or 2048, and ECC, n is up to 192 or 256.

#### Algorithm 1: Montgomery Modular Multiplication

Input: X, Y, M, M', R, where M is modular,  $R=2^n$ ,  $RR^{-1} \pmod{M} = 1$

Output:  $XYR^{-1} \pmod{M}$

Step 1.  $T \leftarrow X \cdot Y$

Step 2.  $S \leftarrow T \cdot M' \pmod{R}$

Step 3.  $U \leftarrow (T + S \cdot M) / R$

Step 4. If  $U \geq M$ , then return  $U - M$ ,

Else return U

The most attractive feature of Montgomery algorithm is it computes modular multiplication without trial division or reversion. It only needs multiplication, addition and right shift which are easy implemented in hardware.

#### 3.2. Improved Montgomery algorithm

The original Montgomery algorithm needs no computing reversion, but when it finishes whole modular multiplication it still computes three long number multiplications. Many researchers has improved Montgomery algorithm to make it adapt to the hardware architecture<sup>[12][13]</sup>. In 1996, KOC proposed an improved algorithm called FIOS, and with

further modification by literature<sup>[14]</sup>, it becomes more convenient algorithm for implementing.

#### Algorithm 2: FIOS Algorithm

D := 0

For j = 0 to e-1 do

(C, S) := X[0]Y[j] + D[0]

U := S·MC (mod  $2^w$ )

(C, S) := (C, S) + M[0]U

(C, S) >>  $\omega$

For i = 1 to e-1 do

(C, S) := (C, S) + X[i]Y[j] + D[i] + M[i]U

D[i-1] := S

(C, S) >>  $\omega$

(C, S) := (C, S) + D[e]

D[e-1] := S

D[e] := C

while (C)

D := D - M

where MC are low  $\omega$  bits in M' which is in Algorithm 1.

In literature<sup>[11]</sup>, pipeline scheme was used to complete the most complex step in the algorithm (C, S) := (C, S) + X[i]Y[j] + D[i] + M[i]U. However, this pipeline mode is not appropriate to compute modular multiplication which operands are not so large, like the usual 256-bit operation in ECC.

We made further modification on this algorithm, so as to accomplish one multiplication during every clock cycle. As we all know if we can accomplish whole modular multiplication in certain number of clock cycles which is exactly the number of multiplications we should compute, we could have a best method to archive what we want. And the number we mentioned above is a constant  $2e^2 + e (e = n / \omega)$ . Under such a circumstance that only one multiplier is employed, the number of clock cycle of Montgomery algorithm need to be  $2e^2 + e (e = n / \omega)$ . For the reason that the data must be prepared before and read out after the multiplication operation, the number of cycle is equal to or less than  $2e^2 + e (e = n / \omega)$ . Hence, when the total number of clock cycle of one Montgomery operation is closer to this value, the performance is better.

The improved algorithm needs more 2 cycles to make things right at the beginning and end of the entire procedure, so the number of clock cycles needed in total is  $2e^2 + e + 2$ . The improved implementation has obvious fewer cycles than algorithm 2. For example, when computing 256-bit modular multiplication, algorithm 2 needs 60 cycles comparing our algorithm 40 cycles, 33.3% less.

#### Algorithm 3: Our Improved Montgomery Algorithm

D := 0

(carrybit, C, S) := 0

```

(C, S) := X[0]·Y[0]
U[0] := S·MC (mod 2ω)
(carrybit, C, S) := (carrybit, C, S) + U[0]M[0]
(carrybit, C, S) >> ω
For j = 1 to e-1 do
  For i = 1 to j do
    (carrybit, C, S) := (carrybit, C, S) + U[i-1]M[j+1-i]

  For i = 0 to j do
    (carrybit, C, S) := (carrybit, C, S) + X[i]Y[j-i]
  U[j] := S·MC (mod 2ω)
  (carrybit, C, S) := (carrybit, C, S) + U[j]M[0]
  (carrybit, C, S) >> ω
For j = e-2 to 0 do
  For i = 0 to j do
    (carrybit, C, S) := (carrybit, C, S) + U[e-1-j+i]M[e-1-i]

  For i = 0 to j do
    (carrybit, C, S) := (carrybit, C, S) + X[e-1-j+i]Y[e-1-i]
  D[e-2-j] := (carrybit, C, S) (mod 2ω)
  (carrybit, C, S) >> ω
D[e-1] := (carrybit, C, S) (mod 2ω)
while ( C )
  D = D - M

```

It can be proved algorithm 3 is equivalent to algorithm 2 in mathematical lever, but the operands sequence is not the same. In the modified algorithm, hardware computes one multiplication every cycle, so it need  $2e^2 + e$  cycles, and adding the cycles of initial and finish step, so it need  $2e^2 + e + 2$  cycles totally.

## 4. Implementation

### 4.1. Hardware Architecture

Based on our modified algorithm, we design the hardware architecture of modular multiplication, see Figure 1. The multiplier is an  $\omega$ -bit C-S structure, the inputs of which are two  $\omega$ -bit multiplicands but output is redundant C-S structure. Since most of multiplication in our algorithm is followed by an addition, the operand of addition can be combined with C-S above. Using his method, we can put multiplication and addition into the same cycle, with spending on the little increment of the delay on critical path.

There are three  $\omega$ -bit registers in modular multiplier, which is used to store the temporary result of multiplication and addition. If the results need to be right shift, store them in the last two registers directly, if no shift needed, store them in the first two registers, so there is no additional cycle for shifting. In fact, the

first two  $\omega$ -bit registers have carry bits, because the result of multiplication and addition maybe exceed  $2\omega$  bits, and so do the adders.

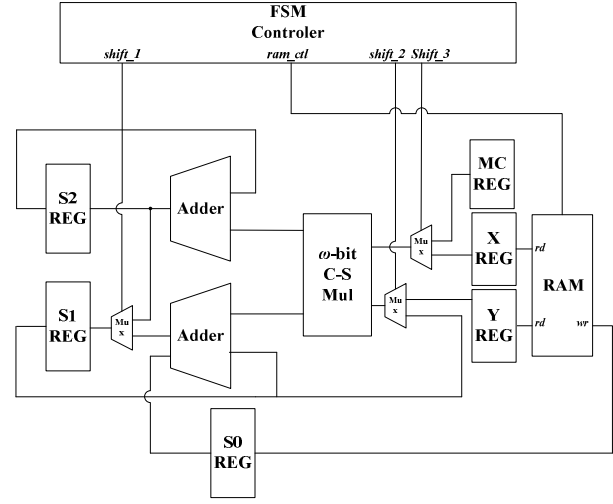


Figure 1. **Hardware architecture**

### 4.2. Computing MM

The proposed implementation is elaborately designed so that  $\omega$ -bit multiplier works in almost every clock cycle during the whole modular multiplication(MM) process. Thus the total cycles of one MM process is nearly  $n/\omega$ , and plus some initial and final necessary cycles the total cycles of proposed MM implementation is  $2e^2 + e + 2$ . The detailed step of computing MM is presented in Table 1.

Table 1. **State of MM implementation**

State	Mul	Add	Write	Read
State <sub>0</sub>				X <sub>0</sub> ,Y <sub>0</sub>
State <sub>1</sub>	X <sub>0</sub> ·Y <sub>0</sub>	S+ X <sub>0</sub> ·Y <sub>0</sub>		MC
State <sub>2</sub>	S <sub>0</sub> ·MC		U <sub>0</sub>	M <sub>0</sub>
State <sub>3</sub>	U <sub>0</sub> ·M <sub>0</sub>	S+ U <sub>0</sub> ·M <sub>0</sub>		M <sub>1</sub>
State <sub>4</sub>	U <sub>0</sub> ·M <sub>1</sub>	S+ U <sub>0</sub> ·M <sub>1</sub>		X <sub>1</sub> ,Y <sub>0</sub>
State <sub>5</sub>	X <sub>1</sub> ·Y <sub>0</sub>	S+ X <sub>1</sub> ·Y <sub>0</sub>		X <sub>0</sub> ,Y <sub>1</sub>
State <sub>6</sub>	X <sub>0</sub> ·Y <sub>1</sub>	S+ X <sub>0</sub> ·Y <sub>1</sub>		MC
State <sub>7</sub>	S <sub>0</sub> ·MC		U <sub>1</sub>	M <sub>0</sub>
...	...	...	...	...
State <sub>17</sub>	X <sub>2</sub> ·Y <sub>1</sub>	S+ X <sub>2</sub> ·Y <sub>1</sub>		X <sub>1</sub> ,Y <sub>2</sub>
State <sub>18</sub>	X <sub>1</sub> ·Y <sub>2</sub>	S+ X <sub>1</sub> ·Y <sub>2</sub>		U <sub>1</sub> ,M <sub>2</sub>
State <sub>19</sub>	U <sub>1</sub> ·M <sub>2</sub>	S+ U <sub>1</sub> ·M <sub>2</sub>		U <sub>2</sub> ,M <sub>2</sub>
State <sub>20</sub>	U <sub>2</sub> ·M <sub>2</sub>	S+ U <sub>2</sub> ·M <sub>2</sub>	RESULT <sub>0</sub>	X <sub>2</sub> ,Y <sub>2</sub>
State <sub>21</sub>	X <sub>2</sub> ·Y <sub>2</sub>	S+ X <sub>2</sub> ·Y <sub>2</sub>	RESULT <sub>1</sub>	
State <sub>22</sub>			RESULT <sub>2</sub>	

## 5. Performance analysis

Our proposed design was coded by Verilog HDL, for  $\omega=64$ , and synthesized by SYNPLIFY 9.2.2. And the whole design is implemented on Xilinx Virtex-II (XC2V250) FPGA. A maximum frequency of 100 MHz could be achieved. Based on this coprocessor, we implemented RSA and ECC Cryptosystems, which can achieve 256-bit point multiplication rate of 172 kbps, and 1024-bit RSA(with CRT) rate of 483 kbps.

Table 2. Performance Modular multiplication

Ref	Tech/device	Freq	Area		Performance ( $\mu$ s)	
			slices	LUTs	1024	256
This work	XC2V250	100	1,331	2,433	5.30	0.38
[21]	TSMC 0.35 $\mu$ m	300	-	-	51*	4.9*
[18]	XC2V3000	161	8,724	-	6.34	-
[19]	XC2VP30	93	3,873	-	44	2.30
[19]	XC2VP30	81	4,233	-	20.4	1.5
[20]	XC2VP30	110.4	4,836	-	-	0.8
[15]	XC2V2000	44.91	5,267	-	23	5.75
[16]	Xilinx Virtex Pro	144	-	5,598	8.10	2.02
[17]	Xilinx Virtex II	135	-	2,593	-	0.40

\*: Equivalent to single modular multiplication by author.

## 6. Conclusion

This paper describes an implementation of modular multiplication coprocessor for both RSA and ECC Cryptosystems. The core module has taken advantage of adopting our improved word by word Montgomery modular multiplication algorithm. In this paper, we have proposed the original creation of the improvement in the modular multiplication algorithm and the hardware realization. The last performance of our coprocessor proves that it stands at a high lever of the reported literature in recent years. And the RSA/ECC Cryptosystems Soc system which using this coprocessor has also reached the high throughput rate.

## 7. References

[1] Rivest R L, Shamir A, Adleman, L. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems" [J], Communications of the ACM, 1978, 21(2): 120-126.

[2] N. Koblitz. "Elliptic Curve Cryptosystems" [J], Mathematics of Computations, 1987, 48:203-209.

[3] V. S. Miller, "Use of elliptic curves in cryptography", in CRYPTO'85, 1986, pp. 417-426.

[4] W. Diffie and M.F. Hellman, "New Direction in Cryptology" [J], IEEE Transactions on Information Theory, 1976, vol22, no.6, pages: 644-654.

[5] A. Bosselaers, R. Govaerts and J. Vandewalle, "Comparison of Three Modular Reduction Functions", CRYPTO'94, 175 ~ 186.

[6] Barrett P, "Implementing the Rivest, Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor" [A]. Advances in Cryptology-Crypto'86 Proceedings [C]. Springer-Verlag, Berlin, 1987, vol 263: 311-323.

[7] P.L. Montgomery, "Modular multiplication without trial division" [J], Math. Computation, 1985, 44:519-521.

[8] H. Sedlak, "The RSA cryptography processor", Proc. of Eurocrypt '87, LNCS 304, Springer-Verlag, pp.95- 105

[9] A. Tenca and C. Koc, "A Scalable Architecture for Modular Multiplication Based on Montgomery's Algorithm", IEEE Trans. Computers, 2003, vol. 52, no. 9:1215-1221.

[10] D. Harris, R. Krishnamurthy and M. Anders, "An improved unified scalable radix-2 Montgomery multiplier"[A]. 17th IEEE Symposium on Computer Arithmetic[C]. 2005.

[11] T. Blum, C. Parr, "Montgomery Modular Exponentiation on reconfigurable hardware", 14th IEEE Symposium on Computer Arithmetic, April 1999.

[12] A. Daly, Marnane and W. Efficient, "Architectures for implementing Montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic"[A], Proceedings of the Tenth ACM International Symposium on Field-Programmable Gate Arrays (FPGA'02)[C], New York: ACM Press,2002.44 -49.

[13] N. Nedjah, Mourelle and Ld. M, "Three hardware architectures for the binary modular exponentiation sequential parallel and systolic", Circuits and Systems I: Regular Papers, IEEE Transactions on Volume 53, Issue 3, March 2006 ,pp: 627 – 633.

[14] Koc C K, Acar T. "Analyzing and comparing Montgomery multiplication algorithm" [J]. 1996, IEEE Micro; 16 (6): 26-33.

[15] F. Crowe, A. Daly and W. Marnane, "A Scalable Dual Mode Arithmetic Unit for Public Key Cryptosystems", Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on Vol.1, pp.568-573.

[16] D. Harris, R. Krishnamurthy and M. Anders, "An Improved Unified Scalable Radix-2 Montgomery

Multiplier”, Proc. of the 17th IEEE Symposium on Computer Arithmetic, June 2005.

[17] K. Kelley and D. Harris, “Parallelized Very High Radix Scalable Montgomery Multipliers”, Conference Record of the Thirty-Ninth Asilomar Conference on Signals, Systems and Computers, October 2005, pp.1196-1200.

[18] R. Garg and R. Vig. “An Efficient Montgomery Multiplication Algorithm and RSA Cryptographic Processor”, International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007), 2007, pp. 188-195.

[19] J. Fan, K. Sakiyama and I. Verbauwhede. “Montgomery Modular Multiplication Algorithm on Multi-Core Systems”,

Singal Processing Systems, 2007 IEEE workshop, 2007, pp.261-266.

[20] K. Sakiyama, B. Preneel and I. Verbauwhede. “A Fast Dalfield Modular Arithmetic Logic Unit and Its Hardware Implementation”, Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS 2006) , 2006, pp. 787-790.

[21] J. Hong, W. Li. “A Novel and Scalable RSA Cryptosystem Based on 32-Bit Modular Multiplier”, 2008 IEEE Computer Society Annual Symposium on VLSI, 2008, pp. 483-486.