

# Implementation of RSA Based on Modified Montgomery Modular Multiplication Algorithm

Gang Feng    Guangsheng Ma    Zhi Yang  
 College of Computer Science and Technology  
 Harbin Engineering Univ.  
 Harbin, China

**Abstract** - A modified Montgomery modular multiplication algorithm using four-to-two CSA (carry save adder) was proposed. The modified algorithm avoids the carry delay and the repeated output/input format conversions. We use this modified algorithm to design the RSA processing unit with 512-bit and 1024-bit key size. The resulting RSA units have a high throughput.

**Keywords:** RSA, Montgomery algorithm, 4-2 CSA

## 1 Introduction

Along with the development of digital communication and electronic business, information security has been more and more demanded. The public-key cryptography has been widely accepted. RSA is a famous public-key cryptosystem, it can effectively realize Digital Signature, Information Authentication and Identity Authentication<sup>[1]</sup>. The central operation of RSA is the modular multiplication of large integers, it can affect the efficiency of the entire system. Montgomery modular multiplication algorithm has a preferable efficiency in the modular exponential operation<sup>[2,3,4]</sup>. It is currently most widely used. We improve the algorithm and present the hardware structure to implement it in this paper. Basing above work, we realize the RSA algorithm. Then we compare the execution time of our algorithm with that of existing methods and analyze the results. Finally our RSA algorithm is described by Verilog HDL and implemented in Virtex2 FPGA.

## 2 Modified Montgomery algorithm

### 2.1 Montgomery algorithm

Montgomery modular multiplication algorithm makes use of add operation and shift operation to avoid the compare operation and subtract operation of traditional division algorithm. Original Montgomery algorithm needs preprocessing and postprocessing to eliminate extra multiplication factor and needs some additional operations to control the bound of the results. To solve these problems, paper[5] improved the original Montgomery algorithm, reduced the procedures of modular multiplication, ensures the output results in (0, N) and the times of loop is still n. The algorithm can be described as:

### Algorithm 1

**inputs:**

module: N is n bits binary expression

multiplier:  $A = (A_{n-1}A_{n-2} \dots A_1A_0)_2$

multiplicand: B is n bits binary integer

**outputs:**

result:  $R = A * B * 2^{-n} \text{ mod } N$

MM (A, B, N)

```
{
  P1: A * B = C = C1 * 2^n + C0;
      P[0] = 0;
  P2: for (i = 1; i < n; i++)
      {
        q_i = (P[i] + C_i) mod 2;
        P[i + 1] = (P[i] + q_i * N + C0) / 2;
      }
  R = P[n] + C1;
  return R;
}
```

Algorithm 1 includes two steps. First, Step P1 computes C ( $C = A * B$ ), C1 and C0 respectively express the high n bits and the low n bits of C. Step P2 computes the product of Montgomery algorithm by C0. The computation of the sum of the three inputs consumes the most time. It is expressed as:

$$P[i + 1] = P[i] + q_i \times N + C_i \quad (1)$$

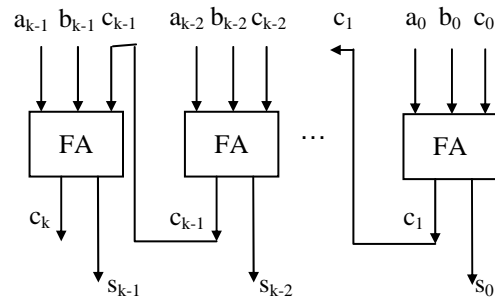


Figure 1. Structure of CSA

Time delay is mainly caused by the carry propagation in the large integers addition. In section 2.2, modified

Montgomery algorithm utilizes CSA (carry save adder) to avoid carry propagation. CSA can be built from Full Adders, the structure is showed in Figure 1.

## 2.2 Montgomery algorithm based on 4-2 CSA

Modified Montgomery algorithm uses 4-2 CSA that is shown in Figure 2. The sum of vector SUM and CARRY equals the sum of X1, X2, X3 and X4. A clock cycle later, the carry save sum of four inputs will arrive output. Algorithm 2 uses this improved technique to describe Montgomery algorithm.

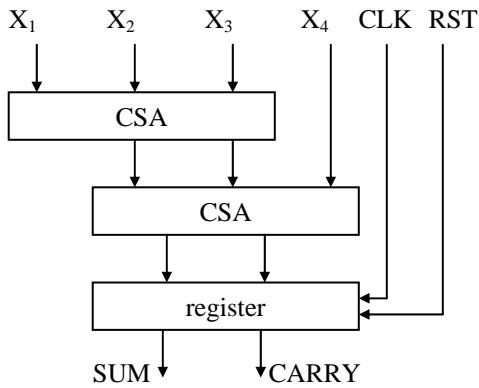


Figure 2. 4-2 CSA

### Algorithm 2

#### inputs:

modular: N is n bits binary expression  
 multiplier: A0, A1 are n bits binary integers  
 multiplicand: B0, B1 are n bits binary integers

#### outputs:

C and S ( $C+S = (A0+A1) * (B0+B1) * 2^{-n} \text{ mod } N$ )

4to2MM (A0, A1, B0, B1, N)

```
{
    C0 = 0;
    S0 = 0;
    C1 = 0;
    S1 = 0;
    Ac = 0;
    for (i = 1; i < n; i++)
    {
        P0: {Ac, As} = Ac + A0i + A1i
        P1: C0, S0 = CSR (S0, C0, As * B0, As * B1);
            T = S0 / 2;
            S0 = S0 / 2;
            C0 = C0 / 2;
        P2: qi = C10 xor S10 xor T;
            C1, S1 = CSR (S1, C1, qi * N, T);
            S1 = S1 / 2;
            C1 = C1 / 2;
    }
}
```

```
C, S = CSR (C0, S0, C1, S1);
return C, S;
```

}

In algorithm 2, CSR expresses 4-2 CSA. Step P0 generates the general form of A from its carry save form A1 and A2; Step P2 computes the product of A and B, B is stored in B0 and B1 in carry save form; Step P2 generates product of Montgomery. P1 and P2 both use 4-2CSA. P0, P1 and P2 can be executed concurrently. The final result is still stored in carry save form. To the 4-2 CSA in P2, the operand T just contains one bit, so we can reduce the loop delay to 2 CSA + 2 XOR + 1 AND. This algorithm is compared with some of previous algorithms in Table 1.

Table1. Delay and Loop times in modular multiplication of RSA

Algorithm	Paper[3]	Paper[6]	Algorithm2
Delay	1 CSA+ 4:1mul	2 CSA+ 4:1mul+ 2 XOR+ 1 AND	1 CSA+ 2 XOR+ 1 AND
Conversion delay	32bits CPA	0	0
Loop times	k+2	k+2	k+2
Critical path delay	32bits CPA	2 CSA+ 4:1mul+ 2 XOR+ 1 AND	1 CSA+ 3 XOR+ 1 AND

Because of the use of CSA, the algorithm effectively eliminates carry propagation, greatly increases the parallel degree and decreases the delay on the critical path. At the same time, modular multiplication result was stored in carry save form, so the conversions from carry save form to single operand form is avoided. Such structure makes RSA algorithm convert the result' form only in the final result computation.

## 3 RSA process unit

### 3.1 RSA algorithm

The encrypt/decrypt of RSA can be formulated as:

Encrypt:  $C = M^e \text{ mod } N$

Decrypt:  $C = M^e \text{ mod } N$  (2)

Where M is Plain text, C is Cipher Text and N is binary module. e and d are respectively encrypt exponent and decrypt exponent,  $e \times d \equiv 1 \text{ mod } (p - 1) \times (q - 1)$ . p and q is two large prime numbers and  $N = p \times q$ .

It's obvious that RSA is a modular exponential operation in essence. The common computing method for

modular exponential operation is square-multiply algorithm, Algorithm 3 shows the encrypt process of RSA.

**Algorithm 3**

**inputs:**

plain text: M is n bits binary integer  
 cipher text:  $E = (1e_{k-2} \dots e_1 e_0)_2$   
 modul: N is n bits binary number  
 constant:  $C = 2^{2n} \bmod N$

**function:**

$MM(A, B, N) = A * B * 2^{-n} \bmod N$

**outputs:**

result:  $M^c \bmod N$

RSA (M, E, N, C)

```
{
  M' = MM (M, C, N);
  R = M' ;
  for (i = k - 2; i >= 0; i --)
  {
    R = MM (R, R, N);
    if (e_i == 1)
      R = MM (R, M' , N);
  }
  R = MM (R, 1 ,N);
  return R ;
}
```

**3.2 Modified RSA algorithm**

Using the modified Montgomery algorithm, we can rewrite Algorithm 3 to realize modified RSA algorithm. This improvement greatly increases throughput, further more. It can be implemented with FPGA or ASIC, etc. Modified RSA algorithm is shown as follows:

**Algorithm 4**

**inputs:**

encrypt: M is n bits binary integer  
 exponent:  $E = (1e_{k-2} \dots e_1 e_0)_2$   
 module: N is n bits binary number  
 constant:  $C = 2^{2n} \bmod N$

**function:**

$4to2MM(A0, A1, B0, B1, N)$   
 $= (A0+A1) * (B0+B1) * 2^{-n} \bmod N$

**outputs:**

result:  $M^c \bmod N$

Modified\_RSA (M, e, N, C)

```
{
  M'1, M'2 = 4to2MM (M, 0, C, 0, N);
  R1 = M'1;
  R2 = M'2;
  for (i = k - 2; i >= 0; i --)
  {
    R1, R2 = 4to2MM (R1, R2, R1, R2, N);
    if (e_i == 1)
      R1, R2 = 4to2MM (R1, R2, M'1, M'2, N);
  }
}
```

$R1', R2' = 4to2MM (R1, R2, 1, 0, N);$   
 $R = R1' + R2';$   
 return R;

}

Table 2 analyses the clock cycle numbers for running the RSA algorithm once, where n is the length of module and k is the length of encrypt exponent.

Table 2. Clock cycle analysis of RSA

Operation	Step	Clock cycle
preprocessing	M'1, M'2	n+2
circulation	R1, R2	1.5(k-1) (n+2)
postprocessing	R1', R2'	n+2
final addition	R	n+2
	total clock number	1.5(k+1) (n+2)

Finally, we describe the RSA algorithm by Verilog HDL and implement the RSA processing unit with 512-bit and 1024-bit key size in Virtex2 FPGA of Xilinx. Encrypt length and decrypt length are respectively 17 and n. Experiment results are listed in Table 3 and Table 4.

Table 3. Results of RSA Encrypt

Device	Key size	Clock cycle (MHz)	TP (Mb/s)	Time (ms)
XC2V3000	512	165.755	6.26	0.08
XC2V6000	1024	157.810	5.83	0.17

Table 4. Results of RSA Decrypt

Device	Key size	Clock cycle (MHz)	TP (Kb/s)	Time (ms)
XC2V3000	512	165.755	219.72	2.3
XC2V6000	1024	157.810	104.90	9.5

**4 Conclusions**

This paper proposed a modified Montgomery modular multiplication algorithm. Basing it, the RSA algorithm is realized and the corresponding circuit structure is presented. The modified algorithm eliminates the carry delay and avoids the form conversions of temporary results by 4-2

CSA during executing large numbers' addition. So the throughput of circuit is increased.

## 5 References

- [1] A. Z. Alkar and R. Sönmez "A hardware version of the RSA using the Montgomery's algorithm with systolic arrays," *Integration*, Vol 38, No. 2, pp. 299-307, Dec. 2004.
- [2] M. Lei, X. Ye, and H. G. Zhang "Montgomery algorithm and its fast implementation," *Computer Engineering*, Vol 29, No. 14, pp. 45-47, July 2003.
- [3] Wang X., Dong W., and Rong M. T. "Implementation of RSA cryptoprocessor based on modified Montgomery modular multiplication algorithm," *Journal of Shanghai Jiaotong University*, Vol 38, No. 2, pp. 240-243, Feb. 2004.
- [4] W. Xiang, L. Guo, and X. F. Bai "An improved scheme & circuit implementation on Montgomery's modular multiplication algorithm," *Computer Engineering and Applications*, Vol 40, No. 36, pp. 115-117, Dec. 2004.
- [5] C. C. Yang, T. S. Chang, and C. W. Jen "A new RSA cryptosystem hardware design based on Montgomery's algorithm," *IEEE Transactions on Circuits and Systems- : Analog and Digital Signal Processing*, Vol 45, No.7, pp. 908-913, July 1998.
- [6] C. McIvor, M. McLoone, and J. V. McCanny "Modified Montgomery modular multiplication and RSA exponentiation techniques," *IEE proceedings of Computers and Digital Techniques*, Vol 151, No. 6, pp. 402-408, Nov. 2004.