COMPARISON OF DIFFERENT IMPLEMENTATION METHODS OF MODULAR EXPONENTIATION OF RSA CRYPTOSYSTEM

Nimmy James[1] and S.Arul Jothi[2]

[1] Student,M.E VLSI Design, Sri Ramakrishna Engineering College, Coimbatore, Tamilnadu,India.

[2] Assistant Professor,ECE Department, Sri Ramakrishna Engineering College, Coimbatore, Tamilnadu,

Abstract **-This paper presents different implementation methods of modular exponentiation of RSA cryptosystem that have been modified to improve performance. It also attempts to perform a comparison based on speed, power and area. It supports multiple key sizes. Therefore it can be easily fit into the different systems requiring different levels of security.**

Key generation, encryption and decryption are the main parts of this project. Encryption and Decryption are done by modular exponentiation.Modular exponentiation is a series of Modular multiplication. In this paper, two schemes are proposed to implement an optimized RSA cryptosystem. The modified Montgomery algorithm made the modular exponentiation simple using one additional multiplication in L-R method or parallel processing in R-L method. As a result, the former minimized the hardware requirement and the latter efficiently reduced the operating time. Each block is coded with VHDL. VHDL code is synthesized and simulated using Xilinx-ISE 10.1.As a result each architecture contributes to speed improvement and area saving.

Keywords- **Modular Multiplication, Modular Exponentiation, Cryptography, Public Key, Private Key**

I.INTRODUCTION

The standard techniques for providing privacy and security in data networks include encryption/decryption algorithms such as Advanced Encryption System (AES) (private-key) and RSA (public-key). RSA is one of the safest standard algorithms, based on public-key, for providing security in networks. Even though the RSA Algorithm is an old and simple encryption technique, there is a scope to improve its performance. The most time consuming process in RSA algorithm is the computation of $M^e \bmod N$ Where M is the text and (e,N) is the key. This paper examines how this computation could be speeded up.

With the increase in data communication and the expansion of internet services like electronic commerce, security problem has become important over the network. To guarantee its security, public key cryptosystem is widely used and RSA is the most popular public key encryption scheme. The implementation consist of three parts: key generation, encryption and decryption process. The key generation stage is used to generate a pair of public key and private key. Figure 1.shows the basic block diagram of public key cryptography. A schematic representation of encryption and decryption are shown in Figure 2

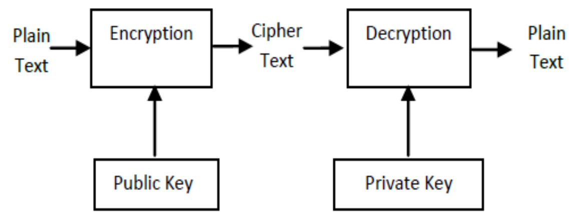

Fig.1.Public Key Cryptography

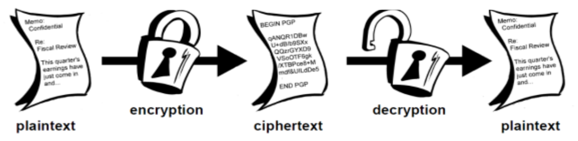

Fig.2.Encryption and Decryption

Although providing high security, currently available RSA hardware needs to be improved on the speed and area issues. The security of the RSA increases as the number of bits in the algorithm increase. However, high number of bits end up with slower architectures and increased area[2]. So the challenge is to provide fast architectures and efficiently used resources as the number of bits increase

The number of research, in parallel with the highly increasing need for the PKC algorithms, had a great jump with the beginning of the 90's. As a result of this jump, outstanding improvements on the time × area product of the RSA implementations came out in the last decade.

While examining the mathematics of the RSA, it was emphasized that the security of the RSA lies in the factorization problem of the large integers. The effort for factorization doubles for every 15-bits when the modulus is about 1024-bits. However, these 15 extra bits require only 5% computation time. Therefore, just speeding-up the operation 5% results in a two times difficult problem of breaking the system. Because of this reason, security is the main reason for the enormous research on speeding-up the RSA algorithm. On the other hand, the speed is also highly needed if RSA is to be used in the real-time applications such as enciphering the real-time audio-video data.

The section II explains an overview of RSA Algorithm. In section III, a modified Montgomery algorithm and two modified exponentiation algorithms are described to make the operation simple. One reduces the hardware resources using the L-R (left to right) binary method, and the other achieves speed improvement using the R-L (right to left) binary method. In section IV, new architectures are proposed for the two methods. Finally,analyzed the performance of the two methods and compare the differences in terms of processing time, power and area.

II.OVERVIEW OF RSA ALGORITHM

RSA is designed by Ron Rivest, Adi Shamir, and Leonard Adleman in 1978. It is one of the best known public key cryptosystems for key exchange or digital signatures or encryption and decryption of blocks of data. RSA uses a variable size encryption and decryption block and a variable size key. It is an asymmetric (public key) cryptosystem based on number theory, which is a block cipher system. It uses two prime numbers to generate the public and private keys. These two different keys are used for encryption and decryption purpose. Sender encrypts the message using Receiver public key and when the message gets transmit to receiver, then receiver can decrypt it using his own private key.RSA operations can be decomposed in three broad steps; key generation, encryption and decryption.The steps of a basic RSA algorithm is shown in figure 3.

Encryption/Decryption

Plaintext block M is encrypted to a Cipher text block C by:

$C = M^e \bmod N$ (1)

The Plaintext block is recovered by:

$M = C^d \bmod N$ (2)

RSA Key Generation

1. Choose two large prime numbers P and Q

2. Compute N=p×q

3.Calculate Φ(N)=(p-1)×(q-1)

4. Select the public exponent e € {1,2,.....,Φ(N)-1}

such that gcd(e, Φ(N))=1

5. Compute the private key d such that d×e≡mod Φ(N)

Fig 3. RSA algorithm

RSA encryption and decryption are mutual inverses and commutative as shown in equation (1) and (2), due to symmetry in modular arithmetic. Hence the encryption engine covers both the operation of Encryption and Decryption.

III.THE MODIFIED ALGORITHM

The modular exponentiation operation is performed as a series of modular multiplications. Hence, the performance of the modular exponentiation depends mainly on the following two criteria:

1) The number of the modular multiplications in the modular exponentiation algorithm.

2) The stand-alone performance and physical area of the modular multiplication module. (The multiplication unit consumes most of the silicon area in an RSA implementation.)

Modular exponentiation operation can further simplified in to series of modular multiplication and squaring operation. This simplification is based on an algorithm known as square and multiply algorithm. This algorithm is based on scanning the bit of the exponent from the left (the most significant bit) to the right (the least significant bit). In every iteration, i.e., for every exponent bit, the current result is squared, if and only if the currently scanned exponent bit has the value 1, a multiplication of the current result by M is executed following the squaring. This is the basic method. Modified modular multiplication modular exponentiation algorithms are discussed below.

A.Modified Modular Multiplication

On the basis of Montgomery algorithm, the RSA system can be designed by the following procedure. The procedure contains three processes, which are mapping, modular exponentiation, and remapping[8]. Each of the processes can be operated by Montgomery modular multiplication. Given the two N- residues A and B, and the extra factor r such that $r=2^n$, Montgomery modular multiplication *REDC(A,B)* is ABr^{-1} mod N, and is depicted as follows:

Algorithm 1.Modified modular multiplication – REDC(A,B)

```
Inputs : $A$, $B$, $N$(modulus)
Step 1 : $R = 0$
Step 2 : for $i = 0$ to $n-1$ do {
    $R = R + A_i B$
    $R = R + R_0 N$
    $R = R/2$ }
Step 3 : if ( R>N ) R = R − N
Step 4 : return ( R )
Output : $R = ABr^{-1}$ mod $N$, $0 \leq R < N$, $r=2^n$
```

Where A, B, and N are n-bit numbers. In this algorithm, the step 3 is an additional reduction to remove the over-size residue. This traditional algorithm[7] limits the range of intermediate output, but it increases complexity. Therefore, the size of r should be two bits larger than that of *N* without the final comparison. Using this modified algorithm, the step 3 can be removed without changing the other steps. In the modified modular exponentiation, the result can be reused as an input for the next modular multiplication after the n+2 iteration steps without bounding the range of the temporary result.

B.Modified Modular Exponentiation

Modular exponentiation is done by repeated modular multiplications. But in RSA cryptosystem based on Montgomery modular multiplication, two extra-processes are needed to perform modular exponentiation. One is mapping to convert input data M into Mr mod N. Then, the output of modular exponentiation becomes M^er mod N. After the results, re-mapping process is performed. It computes REDC (M^er mod N, 1) to remove the extra factor r. Finally the last output results in the desired form , i.e, M^emod N

. First step is to apply L-R (left-to-right) binary method[4]by adopting Montgomery multiplication to modular exponentiation. It is modified to remove the limits due to the first bit of the exponent. The modified modular exponentiation algorithm can be written as follows:

Algorithm 2.Modified L-R binary method

```
Inputs : N(modulus), e(exponent),
M(plaintext),C(constant) = $2^{2(n+2)}$ mod N
Step 1 : M′ = REDC(M, C),
         R = REDC(C, 1)
Step 2 : for i = n-1 to 0 do {
    R = REDC(R, R)
    If( $e_i$=1) then
    {R = REDC(R, M′)}
    else R = R }
```

```
**Step 3 :** R = REDC(R, 1)
**Step 4 :** return (R)
**Output :** R = $M^e$ mod N, $0 \le R < N$
```

When the Montgomery modular exponentiation begins using the LR binary method, the limit due to the first bit of the exponent occurs. If the first bit is 1, select M for the initial input, but if it is 0,select 1. The operation is not fixed and the dependence on input data increases complexity. This problem is solved by fixing the first bit to 1 or by checking the position of the first non-zero bit. But this method requires additional operation. In the modified algorithm, the step 1 – the mapping process - is different from other algorithms. It computes not only M′ but also initial value of R. Thus, the problem is simply solved by using the REDC function once without other operations. It causes slight increase in the number of modular multiplications from 2n+2 to 2n+3 for the worst case. But the small increase can be ignored. Then, R-L (right-to-left) binary method is adopted. The R-L binary algorithm scans the bits of e from the least significant bit to the most significant bit as follows:

Algorithm 3.Modified R-L binary method

```
Inputs : M(plaintext), e(exponent), N(modulus)
C(constant) = $2_{2(n+2)}$ mod N
Step 1 : M′ = REDC(M, C),
         R = REDC(C, 1)
Step 2 : for i = 0 to n-1
2a. If ($e_i$=1) then R = REDC(R, M′)
2b. M′ = REDC(M′, M′)
Step 3 : R = REDC(R, 1)
Step 4 : return (R)
Output : R = Me mod N, $0 \le R < N$
```

The multiplication (step 2a) and squaring (step 2b) operations in the R-L binary method are independent of one another, and thus these steps should be performed in parallel. Provided that two multipliers (one multiplier and one squarer) available, the operation time of the R-L binary method[6] is bounded by the total time required for computing n+2-squaring operations.

As a result, L-R method processor is implemented by only one multiplier, but it takes almost twice longer than the R-L method. On the contrary, R-L method requires only half operating time of the L-R method, but needed two multipliers.

IV.ARCHITECTURE

A.Modular Multiplier – REDC

The REDC module performs the modular reduction by computing the Montgomery multiplication. In hardware implementation every step in Montgomery algorithm is designed by double carry save adder (CSA). The architecture is shown in Figure 4.It consists of two registers for carry and sum, two CSAs, and one CPA (carry propagation adder) to add the carry output and the sum output at every iteration. The CPA performs the last addition and the input register is modified. After the last iteration in REDC module, the two outputs from the carry save adder should be added to get the modular multiplication result.

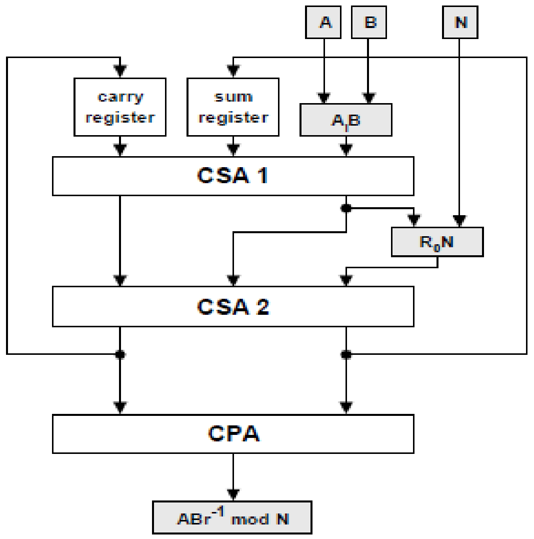

Fig 4.The architecture of REDC module

B.Two Implementation Architectures

Modular exponentiation is executed in three main stages, mapping, exponentiation, and re-mapping. In L-R method design,they are performed by one REDC module, which is used repeatedly. Figure 5 shows the architecture of the 32-bit RSA processor using L-R binary method adopting the modified algorithm.

First, each input is registered. After first multiplication, the temporary output *Mr* mod *N is* stored in M-reg and kept for next iterations. At the end of second multiplication, *1r* mod *N* is computed and registered in rrmodn-reg. During next multiplications, all intermediate outputs are registered in rrmodnreg and after the last iteration – re-mapping process –, we get the desired output from rrmodn-reg.

To perform all these operation using only one REDC module,we need some control blocks. Therefore, the entire system is composed of the REDC module and the control part. The REDC module computes the Montgomery modular multiplication and the control part controls all inputs and outputs of the REDC.

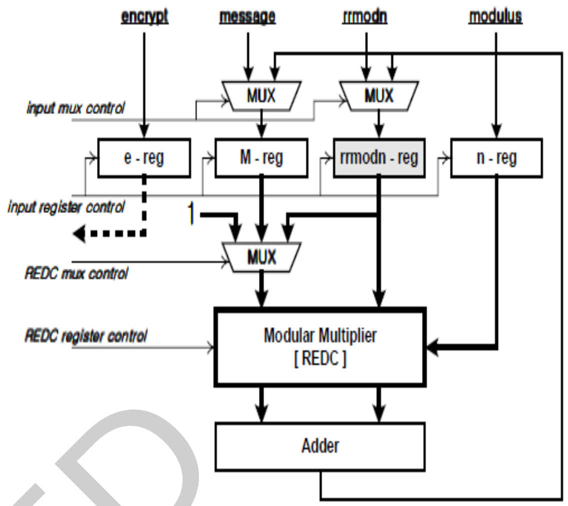

Fig 5.The architecture of RSA Processor based on L-R method

In R-L method, its procedure contains three processes like L-R method. But it is different from L-R method that each process is performed in parallel. Figure 6 shows how the Montgomery multipliers are utilized for modular exponentiations in R-L method.

In mapping stage, the initial values *M* and *1* is multiplied by the mapping factor r. This can be done in parallel using two multipliers. Thus, mapping is done after only one multiplication. At the exponentiation stage, parallel computation begins, so multiplication and squaring is done independently at the same time. During these n-times multiplications, the temporary result is stored in the register. The final Montgomery multiplication is computed at the re-mapping stage to eliminate the mapping factor.

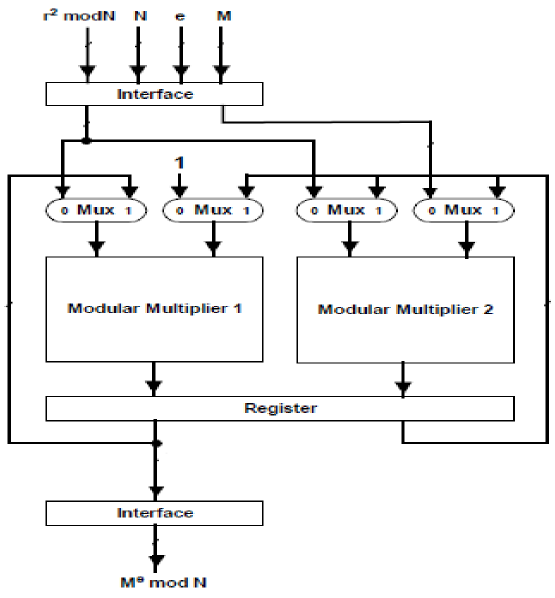

Fig 6.The architecture of RSA processor based on R-L method

As a result, the two implementation architectures show different performances in operating time ,power and chip area. In L-R Method,only one modular multiplier is used to minimize the chip area. On the other hand,faster operation is focused in R-L method than in L-R method. But the two modular multipliers increase chip area.

V.RESULTS AND DISCUSSIONS

A.Simulation Results

Xilinx-ISE 10.1 is used for the simulation. The waveforms are shown below.Figure 7 and figure 8 shows modular exponentiation and encryption and decryption using L-R method.

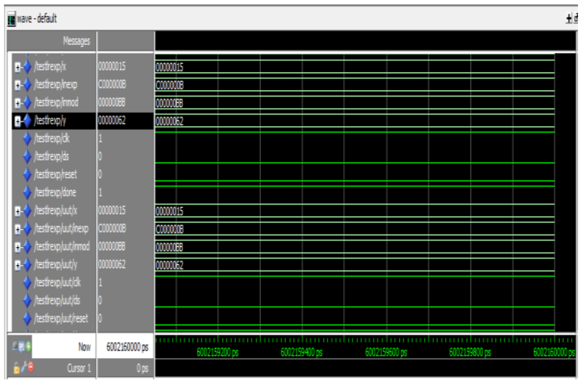

Fig.7.Modular Exponentiation using L-R Method

Fig.8.RSA Encryption and Decryption using L-R Method

The modular exponentiation and RSA encryption and decryption using R-L method is shown in Figure 9 and figure 10 respectively.

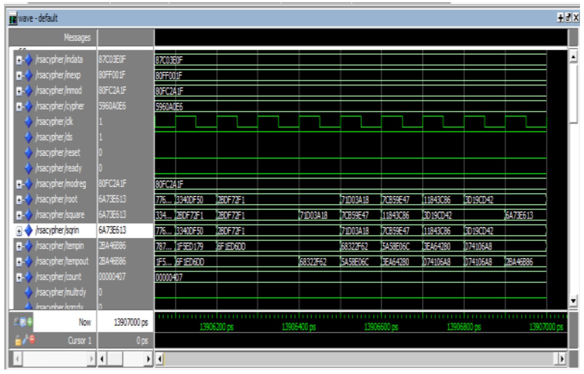

Fig.9.Modular Exponentiation using R-L Method

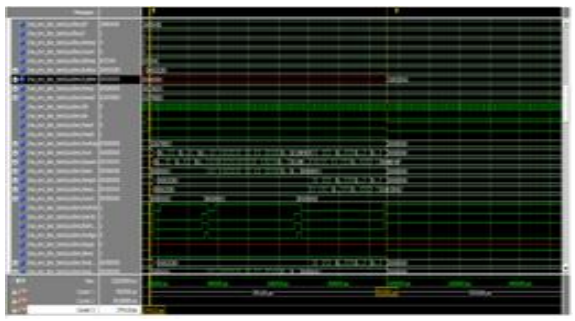

Fig.10.RSA Encryption and Decryption using R-L Method

B.Performance Analysis

Exponentiation Method		L-R Method	R-L Method
Operating Time		2365 ns	785185ps or 959.3 ns
Power Consumption		37 mw	295mW
Device Utilization Summery	Number of Slice Flip flops	107	1423
	Number of 4 input LUTs	251	2648
	Number of Bonded IOBs	36	392
	Number of GCLKs	1	1

Table I. comparison based on Operating time ,Power Consumption and Device Utilization for both L-R and R-L Methods (Encryption /Decryption)

The implementation results includes the design summery from the Xilinx ISE software. Using these results the comparisons in speed, power and area are performed between both L-R and R-L methods. The results are shown in the table I.

V.CONCLUSION

In this paper, two schemes are proposed to implement an optimized RSA processor. The modified Montgomery algorithm made the modular exponentiation simple using one additional multiplication in L-R method or parallel processing in RL method. As a result, the former minimized the hardware requirement and the latter efficiently reduced the operating time. Each proposed method provides a good solution to the practical single chip implementation for large bit size RSA processor.The VHDL code for RSA algorithm is developed block wise. Optimized and synthesizable VHDL code for each block synthesized using Xilinx ISE 10.1

REFERENCES

[1] A.R.Landge and A.H.Ansari, “RSA Algorithm Realization on FPGA”,Volume 2,Issue 7,July 2013,International journal of advanced research in computer engineering & technology

[2] Nasreen J. P., Denila N and Ramola E.P., "A novel architecture for VLSI implementation of RSA cryptosystem real-time QKD", March 2012, IEEE.

[3] Sushanta Kumar Sahu and Manoranjan Pradhan, "FPGA Implementation of RSA Encryption System" Volume 19– No.9, April 2011, International journal of computer applications.

[4] Chiranth E Chakravarthy H.Y.A, Nagamohanareddy P, Umesh T.H, Chethan Kumar M, "Implementation of RSA Cryptosystem Using Verilog", International Journal of Scientific & Engineering Research Volume 2, Issue 5, May-2011 ISSN 2229-5518

[5] Taek-Won Kwon, Chang-Seok You, Won-Seok Heo, Yong-Kyu Kang, and Jun-Rim Choi, "Two Implementation methods of a 1024-Bit RSA Cryptosystem based on Modified Montgomery algorithm" 0-7803-6685-9/01, 2001 IEEE.

[6] Rui He, Jie Gu, Liang Zhang, and Cheng Li, "VHDL for RSA Public Key System"

[7] Nedjah.N and Mourelle L "Two Hardware implementation for the Montgomery Modular Multiplication: Sequential versus Parallel" 2002 IEEE.

[8] Himanshu Thapliyal and M.B Srinivas Sammy, "VLSI Implementation of RSA Encryption System Using Ancient Indian Vedic Mathematics" 1-4244-0549-1/06 ©2006 IEEE.

[9] C. C. Yang, T. S. Chang, and C. W. Jen, “A new RSA cryptosystem hardware design based on Montgomery’s algorithm” IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing, vol. 45, pp. 908-913, July 1998.