Fast and Area Efficient RSA Cryptosystem Design Using Modified Montgomery Multiplication for FPGA Applications

Desiree Juby Vincent

Abstract— RSA(Rivest-Shamir-Adleman) cryptosystem is one of the most widely used public key cryptosystem.The importance of high security and faster implementations paved the way for RSA crypto-accelerators, hardware implementations of the RSA algorithm.The whole RSA includes three parts: key generation, encryption and decryption process. The RSA operation is a modular exponentiation, and its security lies in its inability to efficiently factorize large integers. Basically, the modular exponentiation with a large modulus is usually accomplished by performing repeated modular multiplications, which is considerably time-consuming. As a result, the throughput rate of RSA cryptosystem will entirely dependent on the speed of modular multiplication and the number of performed modular multiplications. To speed up the process of modular multiplication, Montgomery's algorithm is recognized as a very efficient solution, in which it replaces the trial division with a series of additions and division by a power of two. Therefore, it is well suited to hardware implementation and consumes less power and uses smaller amount of space in the FPGA compared to other multiply and reduce methods. This work describes the design of an efficient RSA cryptosystem that uses a modified Montgomery algorithm to increase the speed of modular multiplication and a very fast parallel prefix adder (Kogge- Stone Adder) is employed to reduce the critical path .The design architecture is coded in VHDL, synthesized using Xilinx ISE 12.1 and simulated using Modelsim. Experimental results shows that the modified design obtain the best delay performance compared with the standard design.

Index Terms— FPGA(Field Programmable Gate Array) Kogge- Stone Adder Modular exponentiation Modular multiplication Montgomery algorithm RSA(Rivest-Shamir-Adleman) VHDL(VHSIC hardware description language)

1 INTRODUCTION

IT is widely recognized that security issues play a crucial role in the majority of today's and the future's computer and communication systems. The explosive growth of data communications has made cryptographic algorithms and their implementations a crucial research topic to provide the need of confidentiality, authentication, data integrity, and/or non-repudiation. In order to run successfully, electronic businesses require secure payment channels and digital valid signatures. Cryptography provides a solution to all these problems. Rivest–Shamir–Adleman (RSA) ,is the most widely used public-key cryptosystem, based on the idea originally presented by Diffie and Hellman in 1976. The importance of high security and faster implementations paved the way for RSA crypto-accelerators, hardware implementations of the RSA algorithm.

The RSA operation is a modular exponentiation, and its security lies in its inability to efficiently factorize large integers.Traditional application-specified integrated circuit (ASIC) solutions, however, have the well-known drawback of reduced flexibility and high nonrecurring cost compared with software solutions .The solution, which combines high flexibility with speed and physical security of traditional hardware, is the implementation of cryptographic algorithms on reconfigurable devices as field programmable gate arrays (FPGAs).

• Desiree Juby Vincent is currently pursuing masters degree program in VLSI and Embedded Systems in T.K.M Institute of Technology under CUSAT,Kerala,India.
E-mail: jubyvin@gmail.com

Although computation power has increased with Moore's law, the large increase in computation costs associated with public key cryptosystems has put a significant strain on available computing resources. Thus, there is a growing need for hardware acceleration of public key cryptosystems to reduce the burden of using them. Crypto-accelerators are very promising as they typically achieve better performance and better power efficiency than a software implementation on a generic processor .This project intends to design arithmetic architectures for RSA Cryptosystems which are optimized for modern FPGAs and ASIC technologies. This architecture uses Montgomery algorithm to increase the speed of modular multiplication and the kogge-stone addition (KSA) is employed to reduce the critical path. This Multiplication makes the processing time faster and use comparatively smaller amount of space in the FPGA.

2 SYSTEM ARCHITECTURE

The whole RSA implementation includes three parts: key generation, encryption and decryption process. The key generation stage aims to generate a pair of public key and private key. The cipher text can be decrypted at receiver side by RSA secret key.

A public key encryption scheme has six ingredients:

1.Plaintext: This is the readable message or data that is fed into the algorithm as input.

2.Encryption algorithm: The encryption algorithm performs various transformations on the plaintext.

3.Public and private key: This is a pair of keys that have been

sel
for
4.C
pu
5.D
an

of
er.
Th
the
n)
to

2.1

use
go

Ciphertext: $C=M^e \pmod n$

Decryption

Ciphertext: C

Plaintext: $M=C^d \pmod n$

Key Generation

Select p,q	p,q both prime,p≠q
Calculate n=p×q	
Calculate Φ(n)=(p-1)(q-1)	
Select integer e	gcd(Φ(n),e)=1;1<e<Φ(n)
Calculate d	
Public key	K_U={e,n}
Private key	K_P={d,n}

Encryption

Plaintext:	M<n
Ciphertext:	$C=M^e \pmod n$

Decryption

Ciphertext:	C
Plaintext:	$M=C^d \pmod n$

Fig.1 The RSA algorithm

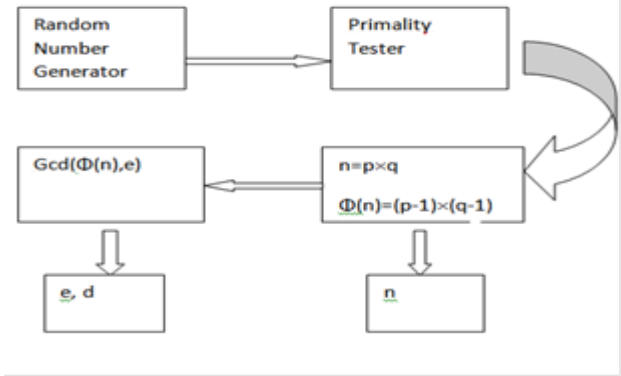

Fig.2 The system architecture for key generation

The system architecture for key generation is shown in Fig.2. A random number generator generates 16-bit pseudo random numbers and the primality tester takes a random number as input and tests if it is a prime .Confirmed primes component pulls out two primes, and calculates n and Φ(n) . N is stored in a register .Φ(n) is sent to the Greatest Common Divider (GCD),where public exponent e is selected such that gcd [Φ(n) ,e] = 1, and private exponent d is obtained by inverting e modulo Φ(n). E and d are also stored in registers. Once n,d, and e are generated, RSA encryption/decryption is simply a modular exponentiation operation. Fig.3 shows the RSA encryption/decryption structure in hardware implementation.

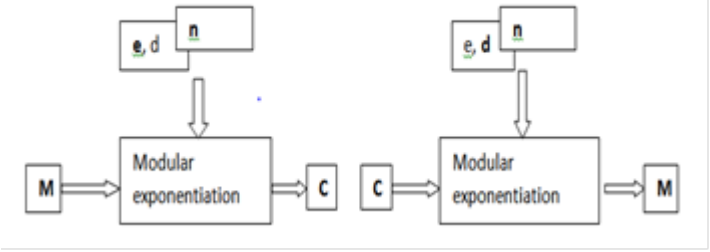

Fig.3 The RSA encryption/decryption structure

The core of the RSA implementation is how efficient the modular arithmetic operations are, which include modular addition, modular subtraction, modular multiplication and modular exponentiation.

1) Modular Multiplication : Modular multiplication can be performed using shift-add multiplication algorithm. Let A and B are two k-bit positive integers, respectively. Let Ai and Bi are the i^{th} bit of A and B, respectively. The algorithm is stated as follows:

Input: A, B, n
Output: M = A*B mod n
P <= A;
M <= 0;
for i = 0 to k-1
if Bi = 1
M <= (M + P) mod n;
end if
P <= 2*P mod n;
end for
return M;

2)Modular Exponentiation: The modular exponentiation operation is simply an exponentiation operation where modular multiplication is intensively performed. Modular exponentiation components can be implemented using LR binary method, where LR stands for the left-to-right scanning direction of the exponent.

The following pseudo code describes the LR binary algorithm. Input: A, B, n
Output: E = AB mod n
E <= 1;

```
for i = k-1 to 0
if Bi = 1
E <= A*E mod n;
end if
if i ≠ 0
E <= E*E mod n;
end if
end for
return E;
```

2.2 THE MONTGOMERY ALGORITHM

One of the widely used algorithms for efficient modular multiplication is the Montgomery's algorithm [4]. This algorithm computes the product of two integers modulo a third one without performing division by M. It yields the reduced product using a series of additions.

Let A, B and M be the multiplicand and multiplier and the modulus respectively and let n be the number of digit in their binary representation, i.e. the radix is 2.

The pre-conditions of the Montgomery algorithm are as follows:

The modulus M needs to be relatively prime to the radix, i.e. there exists no common divisor for M and the radix; The multiplicand and the multiplier need to be smaller than M.

The Montgomery algorithm uses the least significant digit of the accumulating modular partial product to determine the multiple of M to substract. If R is the current modular partial product, then q is chosen so that R+q×M is a multiple of the radix r, and this is right-shifted by r positions, i.e. divided by r for use in the next iteration. So, after n iterations, the result obtained is $R=A\times B\times r^{-n} \bmod M$. A modified version of Montgomery algorithm is given below.

```
Algorithm Montgomery (A, B, M)
Int R = 0;
1: for i= 0 to n-1
2: R = R + ai×B;
3: if r0 = 0 then
4: R = R div 2
5: else
6: R = (R + M) div 2;
return R;
end Montgomery.
```

Fig 4: Montgomery modular algorithm.

In order to yield the right result, we need an extra Montgomery modular multiplication by the constant $r^{2n} \bmod M$. As binary representation of numbers is used, the final result is computed using the following algorithm as given in fig 5.

```
Algorithm Modular Mult(A, B, M, n)
Const C := 2^2n mod M;
Int R := 0;
R := Montgomery(A, B, M);
Return Montgomery(R, C, M);
End Modular Mult.
```

Fig 5: Modular multiplication algorithm

1) Iterative Montgomery Architecture: The interface of the Montgomery modular multiplier is given in Fig 6. It expects the operands A, B and M and it computes $R = (A\times B\times 2^{-n}) \bmod M$.

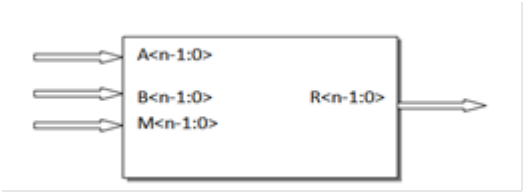

Fig 6: Montgomery multiplier interface

The detailed architecture of the Montgomery modular multiplier is given in Fig 7. It uses two multiplexers, two adders, two shift registers, three registers and a controller .The first multiplexer of the proposed architecture, i.e. MUX21 passes 0 or the content of register B depending on whether bit a0 indicates 0 or 1 respectively. The second multiplexer, i.e. MUX22 passes 0 or the content of register M depending on whether bit r0 indicates 0 or 1 respectively. The first adder, i.e. ADDER1, delivers the sum R + ai× B (line 2 of algorithm of Fig. 4), and the second adder, i.e. ADDER2, yields the sum R + M (line 6 of the same algorithm). The shift register SHIFT REGISTER1 provides the bit ai. In each iteration i of the multiplier, this shift register is right-shifted once so that a0 contains ai.

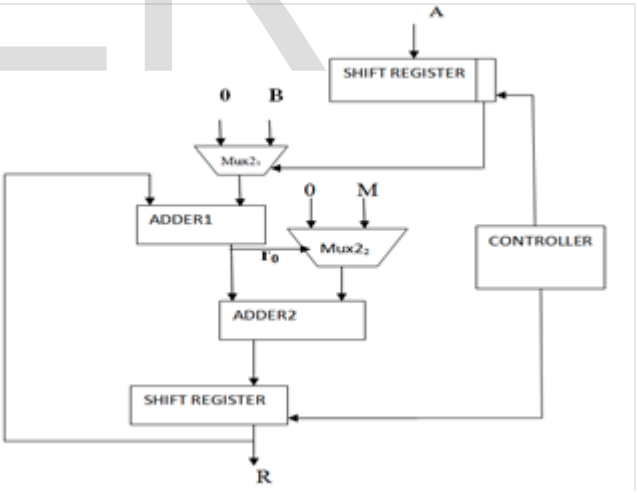

Fig 7: Montgomery multiplier architecture

The role of the controller consists of synchronizing the shifting and loading operations of the SHIFTREGISTER1 and SHIFT REGISTER2. It also controls the number of iterations that have to be performed by the multiplier. For this end, the controller uses a simple down counter. The counter is inherent to the controller. The interface of the controller is given in Fig 8.

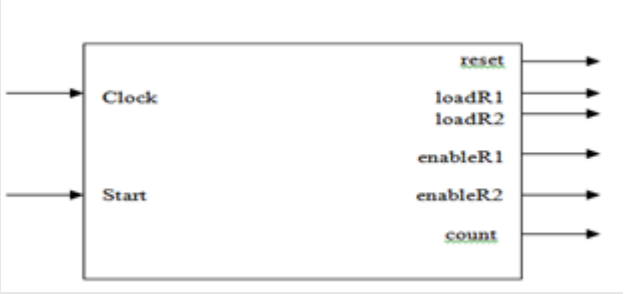

Fig 8: Interface of the Montgomery controller

In order to synchronize the work of the components of the architecture, the controller consists of a state machine, which has 6 states defined as follows:

- S0: Initialize of the state machine; Go to S1;
- S1: Load multiplicand and modulus into the corresponding registers; Load multiplier into shift register1;Go to S2;
- S2: Wait for ADDER1;Wait for ADDER2;Load multiplier into shift register2;Increment counter; Go to S3;
- S3: Enable shift register2;Enable shift register1;
- S4: Check the counter; If 0 then go to S5 else go to S2;
- S5: Halt;

2) Modular Multiplier Architecture: The modular multiplier yields the actual value of A×B mod M. It first computes R = A×B×2^{-n} mod M using the Montgomery modular multiplier. Then, it computes R × C mod M, where C = 2^{2n} mod M.

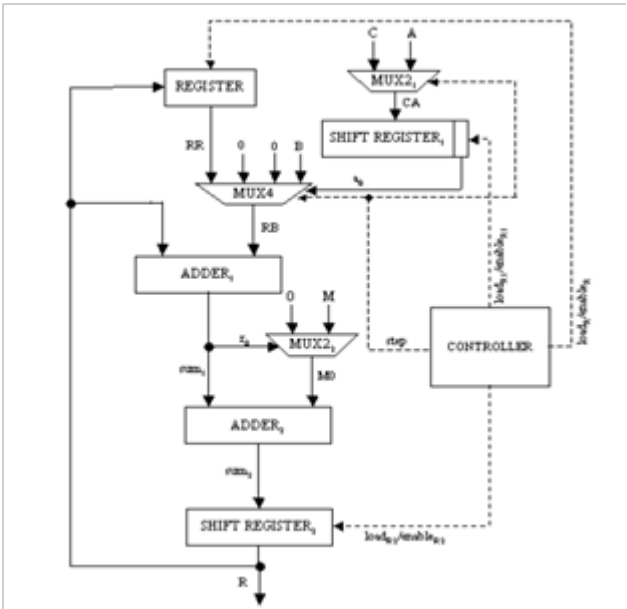

Figure 9: The modular multiplier architecture

The modular multiplier uses a 4-to-1 multiplexer MUX4 and a register REGISTER.

- Step 0: Multiplexer MUX4 passes 0 or B. MUX2 passes A. It yields R1 = A×B×2^{-n} mod M. The register denoted by REGISTER contains 0.
- Step 1: Multiplexer MUX4 passes 0 or R. MUX2 passes C. It yields R = R1×C mod M. The register denoted by REGISTER contains the result of the first step computation, i.e. R = A×B×2^{-n} mod M.

The modular multiplier controller does all the control that the Montgomery modular multiplier needs as described in the previous section. Furthermore, it controls the changing from step 0 to step 1, the loading of the register denoted by REGISTER.

2.3 The Modified Montgomery Algorithm

A modified version of Montgomery multiplication can be obtained using the following algorithm.

```
P:=0;
For i in 0 to k-1 loop
q(i):=(p(0)+x(i)*y(0))mod 2;
p:=(p+x(i)*y+q(i)*m)/2;
end loop;
if p>=m then z:=p-m; else z:=p; end if;
```

In the above algorithm the critical path includes q(i) and p computation. An alternative algorithm which precomputes q(i+1);ie, q for the next iteration is given below.Its advantage in hardware implementation is that p and q(i) can be computed in parallel.

```
P:=0; q(0):=x(0)*y(0);
For i in 0 to k-1 loop
q(i+1):=((p(1:0)+x(i)*y(1:0)  + q(i)*m(1:0))/2
+ x(i+1)*y(0))mod 2;
p:=(p+x(i)*y+q(i)*m)/2;
end loop;
if p>=m then z:=p-m; else z:=p; end if;
```

The carry propagation delay in the above algorithm can be reduced using kogge-stone adder. KSA is a parallel prefix form carry look ahead adder. It generates carry in O (logn) time. In KSA, carries are computed fast by computing them in parallel.

The complete functioning of KSA can be easily comprehended by analyzing it in terms of three distinct parts :

1. Pre processing

This step involves computation of generate and propagate signals corresponding to each pair of bits in A and B. These signals are given by the logic equations below:

pi = Ai xor Bi
gi = Ai and Bi

2. Carry look ahead network

This block differentiates KSA from other adders and is the main force behind its high performance. This step involves computation of carries corresponding to each bit. It uses group propagate and generate as intermediate signals which are given by the logic equations below:

$P_i = P_i$ and P_{iprev}

$G_i = G_i$ or (P_i and G_{iprev})

3. Post processing

This is the final step and is common to all adders of this family (carry look ahead). It involves computation of sum bits. Sum bits are computed by the logic given below:

$S_i = P_i$ xor C_{i-1}

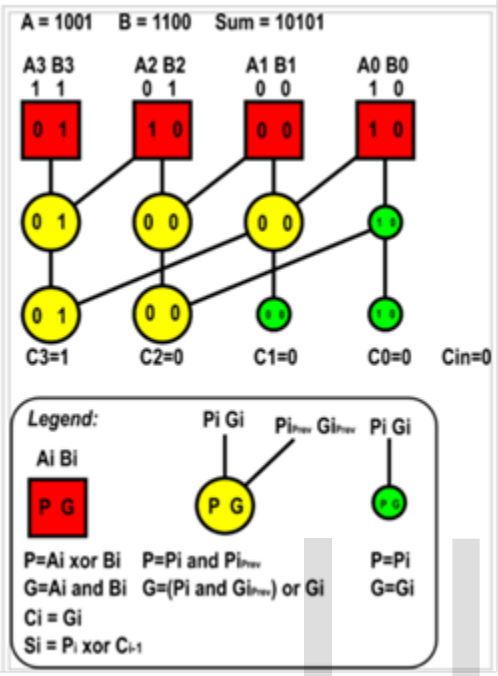

Fig10.illustration of kogge-stone adder

3 RESULTS AND DISCUSSIONS

The modules are modeled using VHDL in Xilinx ISE Design Suite 12.1 and the simulation of the design is performed using Modelsim SE 6.2c to verify the functionality of the design.

A. RSA key generation

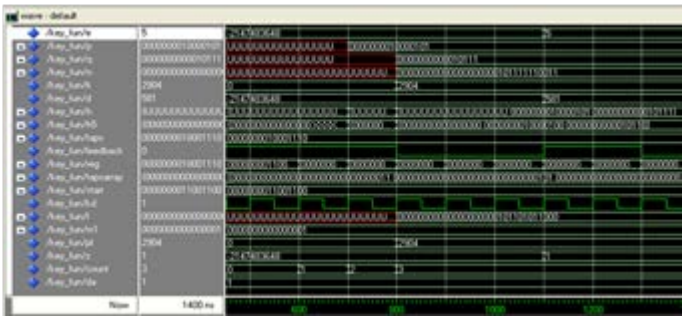

Fig11.1 Simulation result of RSA key generation

RSA key generation obtains the value of public key e and private key d. The value of prime numbers are obtained from LFSR. Then values of n and Φ(n) is calculated. Value of e is calculated such that gcd (e,Φ(n)) is 1.The value of p and q is obtained as 133 and 23 respectively. N=p*q=133*23=3059,Φ=132*22=2904.The value of e is selected as 5.The value of d is obtained as 581.

B. Encryption

A 32 bit data is given as message input.Then e and n values are given as inputs. Inputs given are indata=73,inexp=5 and inmod=3059.Output cypher =2588.

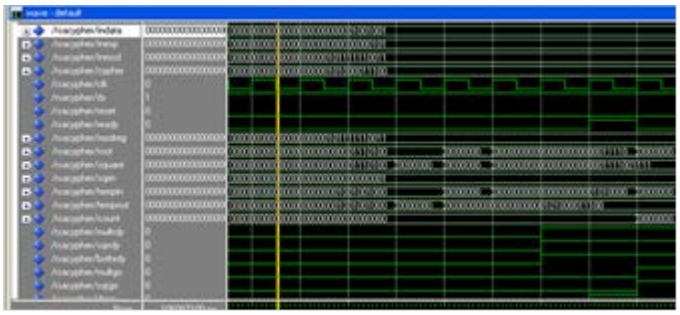

Fig 11.2 Simulation result of encryption

C. RSA block

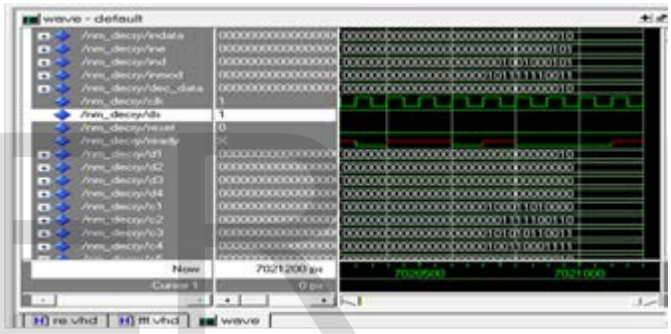

Fig 11.3 Simulation result of RSA block

It takes a message value as input and calculates the value of cipher text. Then message value is decrypted.using the RSA decryption equation, $M=C^d(\text{mod } n)$.

D. Montgomery multiplier

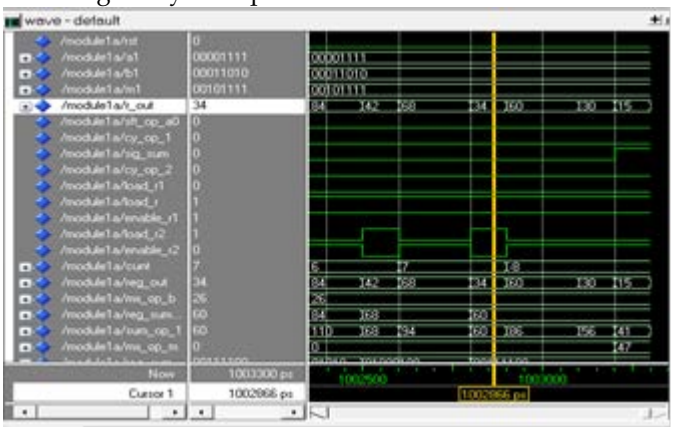

Fig 11.4: The modular multiplier behavior during the first multiplication: Montgomery(15, 26, 47) = 34

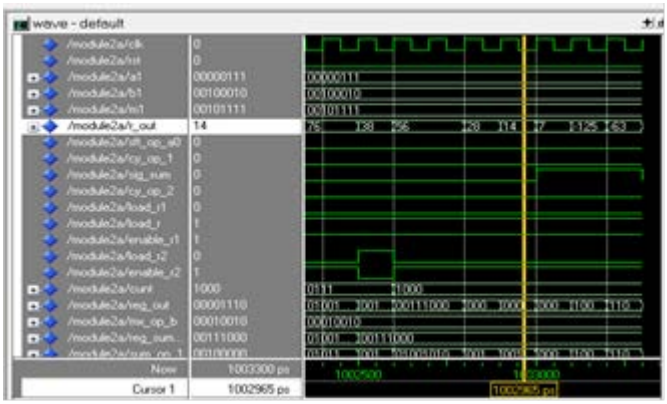

Fig 11.5: The modular multiplier behavior during the second multiplication: Montgomery(7,34,47) = 14

The Montgomery modular multiplier prototype for operands A = 15,B = 26, M = 47 and so the constant C = 2^{2x6} mod 47,which is C = 7,is simulated.

TABLE I
MINIMUM PERIOD AND MAXIMUM FREQUENCY DEPENDING ON THE BIT LENGTH.

Method	Operand size	Min.Period	Max.Freq
Add and Shift	32	11.598ns	86.221MHz
	128	17.292ns	57.829MHz
	512	42.026ns	23.795MHz
Modified Montgomery	32	1.878ns	532.48MHz
	128	2.150ns	465.12MHz
	512	3.298ns	303.21MHz

TABLE II
FPGA CONFIGURABLE LOGIC BLOCK SLICES' USAGE (OUT OF 4656) DEPENDING ON THE BIT LENGTH.

Method	Operand size	No.of Slices used	Percentage utilization
Add and Shift	32	218	4%
	128	782	16%
	512	3324	71%
Modified Montgomery	32	185	3%
	128	654	11%
	512	2434	52%

Implementation results of add and shift architecture and Montgomery architecture,which were implemented on a Xilinx FPGA XC3S500E-4FG320,are shown in the following tables. Table 1 compares the timing operation, showing the maximum workable frequencies and minimum workable periods; Table 2 compares the device utilization by the number of used FPGA configurable logic block slices and its representative percentage.These results shows that operating frequency of modified Montgomery multiplication in fast compared to add and shift method.Critical path is reduced using KSA.For Montgomery method device utilization is also less.

4 Conclusion

This project introduces the design of an efficient RSA cryptosystem that uses Montgomery algorithm for modular multiplication.This architecture speed up modular multiplication and also eliminates the need for memories by checking for primality and selects two prime numbers simultaneously while the random numbers were being generated.RSA key generation, encryption and decryption using shift and add modular multiplication method and Montgomery multiplication has been designed using VHDL and simulated using modelsim. Montgomery replaces the division by adding a shift and modulate, which is much faster than multiply and reduce method .The use of modified Montgomery and kogge-stone adder reduces the critical path and response time.A comparative study in terms of area and speed of Montgomery multiplication and add and shift multiplication is also done.

References

[1] Sushanta Kumar Sahu & Manoranjan Pradhan, "Implementation of Modular multiplication for RSA Algorithm" ,IEEE conference on communication systems and network technologies 112–114,2011

[2] Jainath Nasreen.p, Denila.N, "A Novel Architecture for VLSI Implementation of RSA Cryptosystem", IEEE International conference on ICCEET 606–609,2012

[3] Gustavo D. Sutter, Member, IEEE, Jean-Pierre Deschamps, and José Luis Imaña "Modular Multiplication and Exponentiation Architectures for Fast RSA Cryptosystem.Based on Digit Serial Computation."IEEE Transactionson industrial electronics, vol. 58, NO. 7, JULY 2011

[4] P. L.Montgomery, "Modular multiplication without trial division, "Math.Comput., vol. 44,no.170,pp.519–521, Apr. 1985.

[5] Douglas L. Perry, "VHDL Programming by Example, "Fourth Edition, pp.277,TataMcGra-Hill Publishers Ltd

[6] William Stallings, "Cryptography and Network Security: Principles and Practices." 3rd edition.

[7] Aaron E. Cohen and Keshab K. Parhi,"Architecture Optimizations for the RSA Public Key Cryptosystem: A Tutorial," IEEE circuits and systems magazine, fourth quarter 2011

[8] David NarhAmanor,"Efficient Hardware Architectures for Modular Multiplication"The University of Applied Sciences Offenburg, Germany,2005

[9] N. Nedjah and L. Mourelle."A review of modular multiplication methods and respective hardware implementations".Informatica, 30 :111–130, 2006

[10] C. Mclvor, M. McLoone, and J. V. McCanny, "Fast Montgomery modular multiplication and RSA cryptographic processor architectures," inConf. Rec. 37th Asilomar Conf. Signals, Syst., Comput., 2003, vol. 1,pp. 379–384,1975.

[11] A. Miyamoto, N. Homma, T. Aoki, and A. Satoh, "Systematic design of RSA processors based on high-radix montgomery multipliers," IEEETrans.VLSI,pp.1–11,2011L. Hubert and P. Arabie, "Comparing Partitions," *J. Classification*, vol. 2, no. 4, pp. 193-218, Apr. 1985. (Journal or magazine citation)

[12] R.J. Vidmar, "On the Use of Atmospheric Plasmas as Electromagnetic Reflectors," *IEEE Trans. Plasma Science*, vol. 21, no. 3, pp. 876-880, available at http://www.halcyon.com/pub/journals/21ps03-vidmar, Aug. 1992. (URL for Transaction, journal, or magzine)

[13] J.M.P. Martinez, R.B. Llavori, M.J.A. Cabo, and T.B. Pedersen, "Integrating Data Warehouses with Web Data: A Survey," *IEEE Trans.*

Knowledge and Data Eng., preprint, 21 Dec. 2007, doi:10.1109/TKDE.2007.190746.(PrePrint)