Introducing Primality Testing Algorithm with an Implementation on 64 bits RSA Encryption Using Verilog

*Rehan Shams and**Fozia Hanif Khan,***Umair Jillani and****M. Umair

***Abstract*---A new structure to develop 64-bit RSA encryption engine on FPGA is being presented in this paper that can be used as a standard device in the secured communication system. The RSA algorithm has three parts i.e. key generation, encryption and decryption. This procedure also requires random generation of prime numbers, therefore, we are proposing an efficient fast Primality testing algorithm to meet the requirement for generating the key in RSA algorithm. We use right-to-left-binary method for the exponent calculation. This reduces the number of cycles enhancing the performance of the system and reducing the area usage of the FPGA. These blocks are coded in Verilog and are synthesized and simulated in Xilinx 13.2 design suit.**

Index terms- RSA, Verilog, Cryptosystem, Primality Testing, Decryption, Encryption, Implementation, Key Generation, Modular Exponentiation

I. INTRODUCTION

It is very important in today's world to develop new ways to guarantee their security as far as the data communication is concerned. There has been a lot of work going on in the field of cryptography and in the recent years it has increased exponentially. As the utilization of communication system increases so does the need for securing is sufficient. Many algorithms are designed to meet these needs. Cryptographic algorithms have two major types: symmetric and asymmetric [1].

Symmetric cryptography requires sharing of a single key at both ends. The main problem is the selection of the key privately. In asymmetric (public key) cryptography this problem is overcome by using an algorithm that deals with two keys. One key is for encryption and the other one is to decrypt the same message. The idea behind publishing one key (the public key) and keeping the other one secret (the private key) can surely make the whole procedure more secure and protected. Only those will be able to read the message who may also have the private key as well, if someone wants to encrypt the message then it is necessary to have both keys [2]. RSA algorithm belongs to this type of cryptography.

The problem discussed in many ways [12][13][14]. [12] has provided the high speed RSA implementation of FPGA platforms, [13] showed the high speed RSA implementation of a public key block cipher-MQQ for FPGA platforms , also [14] has provided the implementation of RSA algorithm on FPGA. This paper aims to extend the work done by [14]. Here we are implementing our proposed engine for 64 bits RSA encryption. For RSA implementation the most important step is to select the prime number, therefore, in this paper we are proposing a fast Primality testing algorithm that supports in generating the key which is the first step of RSA algorithm [15]. Also other algorithms like LFSR, Miller Rabin, Extended Euclidean and Modular Exponentiation have been successfully implemented by using the proposed technique of XILINX ISE 13.2.

As far as the significance of the RSA is concerned it can be used as a tool for exchanging the secret information such as messages and conversation by generating the keys and producing digital signatures. However, the complexity comes from calculating the prime factors of large numbers. The work presented in this paper implements the modular exponentiation operation by simple right-to-left-binary method, which helps to reduce the processing time.

II. OVERVIEW OF RSA

The RSA algorithm was proposed by Rivest, Shamir and Adleman of MIT in 1977. This method is known to be the best and commonly used as a public-key scheme which is based on exponentiation in a finite (Galois) field over integers modulo a prime. The security is basically due to cost of factoring large numbers. As already defined in RSA cryptosystem there are two keys, the public and the private key. The public key is known to the world and the private key is supposed to be kept secret. Therefore, an anonymous person will not be able to decrypt the encrypted message if he does not have the private key. The safety depends upon the length of the key, longer the key-length much safer is the data [3].

III. PROPOSED PRIMALITY TESTING ALGORITHM

As far as the significance of the proposed algorithm is concerned, this algorithm is very useful in RSA algorithm for finding the prime numbers and different modification of cryptosystem. Primality testing is one of the fundamental requirements of many cryptographic algorithms and this algorithm is simply another contribution toward the Primality testing techniques.

*Department of Telecommunication, Sir Syed University of Engineering and Technology, r.shams@hotmail.com,
**Department of Mathematics, Sir Syed University of Engineering and Technology ms_khans2011@hotmail.com,
***Department of Telecommunication, Sir Syed University of Engineering and Technology ujilani@gmail.com,
****Department of Electronics, umairssuet@hotmail.com.

A. Steps of algorithm

i. Choose a number N as an integer [2, N-1]

ii. Check if $(\frac{N-1}{2})$ is an integer then the number is odd or prime the number is even.

iii. Again check the condition for any integer "r" which is greater than 1, the square root of $r^{N-1} \equiv 1 \pmod N$ by choosing any integer a. If the remainder is not 1 then N may be composite.

iv. Pick a random number b>N, let x be any positive integer. If, $b^{\frac{x}{2}} \equiv -1(mod N)$ then again pick the random number or, If $\gcd(b^{\mp\frac{x}{2}}, N)$ is nontrivial factor of N, we are done.

v. Again check for any integer "S" according to the Euclid's theorem, that is N and s are relatively prime if $s^{N-1} \equiv 1 \pmod N$, by taking φ(N)= N-1orchecking the prime number if above condition fails then there must be some factor of N and that can be found by Applying the factor algorithm for finding the factor of N, if possible.

IV. MAIN STEPS OF RSA ALGORITHM

Following are the steps involved in the RSA algorithm:

A. Key generation

Key generation is the most important aspect of RSA Algorithm. According to the procedure the encryption key e is available but the decryption key d is not known to all. Mathematically this procedure is defined as, M is the actual message, C is the converted message or cipher text by using publicly available encryption key e, and d is the decryption key.

$$C = M^e \pmod m$$

$$M = C^d \pmod m$$

RSA encryption and decryption are mutual inverses and commutative [4]. RSA algorithm is divided into blocks and each block is then implemented. Generation of public and private keys is the first step which is summarized in Figure1.

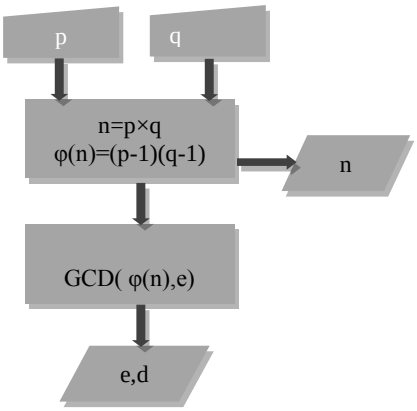

Figure.1 RSA key generation

It begins with a pseudorandom number generator that generates 32-bit pseudo numbers. These pseudorandom numbers are stored in a FIFO. When the FIFO is full the pseudorandom number generator will stop working. Random numbers from the FIFO are pulled out by the Primality tester as this happens, the PRNG will start again to make sure that the FIFO remains filled. Coming back to the Primality tester, it takes a random number as input and check for the number to be prime. If the number is proved as prime, it goes to the Prime FIFO. When the prime FIFO is not full, the Primality tester only pulls a number out of the FIFO. When there is a need of new keys, two random prime numbers are extracted from the prime FIFO. The structure is shown in Figure. 2.

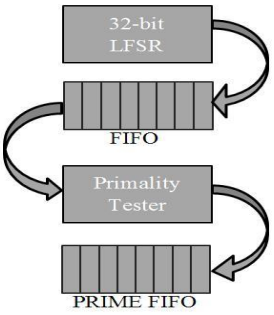

Figure.2 Prime Random Number Generation

These two prime numbers are used to calculate n and Φ(n). Φ(n) is forwarded to the Greatest Common Divider (GCD) calculator where a number e is selected which will be the public or encryption key if it satisfies the condition that GCD(Φ(n), e) = 1. This will show that modular inverse d of this number exists and the modular multiplicative inverse will be considered as private or decryption key. We got e, d and n, for encryption and decryption. Modular exponentiation is applied to encrypt or decrypt the data. This is something that has to be focused because the performance of RSA algorithm depends on how modular arithmetic functions are calculated. They are the core of the algorithm. This process is shown in Figure 3:

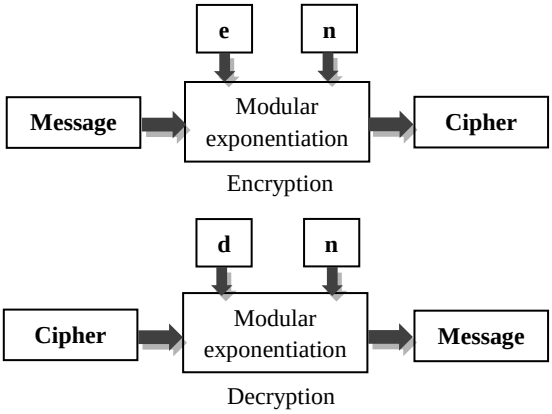

Figure3: Encryption and Decryption

B. Random Number Generator

Linear Feedback Shift Register (LFSR) is the most useful technique among all the available techniques for generating pseudorandom numbers. Here 32-bit pseudorandom numbers are generated using LFSR. LFSR generates a periodic sequence such that the pattern in which numbers are generated will be repeated after certain interval. By using primitive polynomial, maximum length of LFSR sequence is 2n-1. A 32-bit LFSR will produce a sequence of over 4 billion random bits, or 500 million random bytes. The polynomial used for generating this sequence of 32-bit pseudorandom numbers is as under.

$$P(x) = x32+x22 +x2+x1 +1 \quad (1)$$

C. Primality Tester

The numbers generated by the LFSR consist of both prime and composite numbers. One of the most widely used and fastest Primality testing algorithms that produce good result in a polynomial time is the Miller-Rabin test [5]. The main purpose of this test is to separate the prime numbers and then save them for further use. We eliminate the even numbers generated from the LFSR and feed the odd numbers to the Primality tester [6].

The odd integer n from the LFSR to be tested, k determines the numbers of round for the test. For each round, we use a new integer 'a', the upper and the lower bound for the integer are 2 and n-2 respectively, which may be 1 or 0 for the composite or probable prime numbers. If the result of any round is 0 (composite), then there is a possibility that n is composite and we will not move any further. Since there is high probability that n is prime, if the result of all the rounds is 1 (probable prime), then it's very rare that probable prime n might be composite [7].

D. GCD

Two prime numbers are required from the Prime FIFO for the generation of keys. Applying respective operations on these two primes give n and Φ(n). After Φ(n), select a number e which follows the condition GCD(Φ(n),e) = 1, which means that e is relatively prime to Φ(n). This will prove that there is an existence of modulo inverse of e.

$$e \times d \bmod(\Phi(n)) = 1 \quad (2)$$

Here we implement the extended Euclidian algorithm for this purpose. When the GCD is 1, the module returns the values of e and the modular multiplicative inverse is d. Otherwise, e gets an increment of 2 and GCD is calculated again, this continues until the value of e satisfies the condition and a positive inverse is found. e will be used as the encryption key and d as the decryption key. The steps of Extended Euclidian algorithm are as follows:

```
extended_euclidean_main(p,q)
   e =1
   (gcd, d) = (0, 0)
while (gcd != 1 || d < 0)
begin
      e = e + 2
      (gcd, d)= extended_euclidean_loop((p – 1)(q – 1), e)
end
return (e,d)

extended_euclidean_loop(a,b)
   (y, y_prev) = (1, 0)
while b != 0
begin
      (y, y_prev) = (y_prev – a/b*y, y)
      (a, b) = (b, a mod b)
end
return (a, y_prev) {gcd( (Φ(n), e) is a, inverse is y_prev}
```

E. Encryption/Decryption

The process of encryption is converting plain text in such a way that eavesdroppers or hackers cannot read that text called Cipher text. Decryption is the inverse process by which cipher text is converted back into the form that is readable namely plain text. After the keys generation, RSA encryption and decryption is done using the mathematical operation $C = M^e \pmod n$ and $M = C^d \pmod n$ respectively. Hence encryption/decryption is just a modular exponentiation operation like modular addition, modular subtraction and modular multiplication.

F. Modular Exponentiation Operation

Modular exponentiation operation is simplified using square and multiply algorithm. It is done by using right-to-left-binary method. Binary method calculates M X e by using the binary expression of exponent e. In this method the exponentiation operation is broken into a series of squaring and multiplication. The purpose of using the binary method is to speed up the exponentiation calculation. The LSB binary exponentiation algorithm (also called as right-to-left binary exponentiation

algorithm), starting from the least significant bit position and calculates the exponent e and proceeds towards left, which can be write as follows [8]:

Input: M, e, n
Output: C = M_e mod n
Let e contain k bits
If e_{k-1}=1 then C=M else C=1
For i=0 to k-1
C=C×C
If e_i=1 then C=C×M

This algorithm works on the logic of scanning bits from the right for 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[9].

V. HARDWARE IMPLEMENTATION

For the implementation on hardware design, the RSA Cryptosystem is divided into 4 modules:

A. Initial module
B. Modular exponentiation
C. Core algorithm
D. Top module

A. Initial Module

For generating random numbers for the algorithm this module consists of a 32-bit LFSR, which are then stored in the FIFO if they are proven to be odd. As this FIFO fills completely, the Miller Rabin Primality tester takes a number out from the FIFO and test it for prime. The exponentiation part of this algorithm is done by using the right to left binary algorithm implemented for encryption/decryption. If the result of test is positive, then the number is stored in PRIME FIFO which will be utilized later by the algorithm. This process will only stop when the PRIME FIFO is full.

B. Modular Exponentiation

Calculating the modular exponentiation of a number is the most important and time consuming part of RSA algorithm. For this purpose we implement the Square and Multiply algorithm by using the right-to-left-binary approach. It speeds up the exponent calculation and limits the number of cycles needed. This exponent function is also required in Miller and Rabin tester so that module can be called for further procedure and calculation, saving both space and time.

C. Core Algorithm

Here we implement the basic functions and steps of RSA algorithm. This is further divided into two steps: Key generation and Cryptography.

When a new user comes to the system this module takes two numbers as input. These numbers should be 32-bit prime, n and Φ(n) are calculated by inserting them into the multiplier, hence getting a 64-bit number. Φ(n) is then used to find the encryption and decryption keys and the Euclidean algorithm calculates the GCD of Φ(n) and an odd number. If the GCD is found to be 1 this shows that the modular inverse of the odd number exists and the number is co-prime. This number is stored in a register and will be used as the encryption key. Furthermore, extended Euclidean algorithm is used to find the decryption key which is the modular inverse of the encryption key. Hence, key generation procedure is completed for the user and who is allotted with a public and private key that will be used to encrypt and decrypt the data. The Modular Exponentiation is applied to encrypt and decrypt the messages. Register e_d switches the exponent value so that it could be used for both encryption and decryption by just changing the exponent value to the respective key.

D. Top Module

For maintaining the RSA flow, top module controls the functions of the other modules and interconnects them. This module implements a controller with multiple checks so as to get the desired results.

Table 1: Frequency table by proposed engine

Word size [bits]	Clock frequency
128	65.532
136	68.31
160	70.763
208	72.546
288	81.325

From the above table we can see that the frequency can be decreased by the proposed encryption engine. There is a falling rate of 15% in each case of word size. Table 2 shows the simulated results in which we consider the maximum period 11.47ns with maximum frequency 59 MHz by taking maximum combinatorial Path delay 11ns and the maximum output required time is clock 8.457.

Table 2: Simulation Results

Logic utilization	used	Available	utilization
No. of slice	432	587	73%
Number of slice flip plops	7632	9512	80%
No. of 4 inputs LUTs	568	931	61%
No of bonded IOBs	105	323	32%
No. of BRAMs	8	20	40%
No. of MLT 18X18SIOs	15	20	75%
No. of GCLKs	6	24	25%

VI. SIMULATION RESULTS

LFSR, Miller Rabin, Extended Euclidean and modular exponentiation have been successfully written and tested on Xilinx ISE 13.2. The simulation shows the desired results of these algorithms. Following section shows the simulation results of these algorithms.

A. Linear Feedback Shift Register

The pseudo random numbers are generated by a 32-bit Linear Feedback Shift Register which is simulated in Xilinx ISE.

Figure 4: Simulated Waveform for Pseudo-Random number generator

Figure 4 shows some 32-bit random numbers that are generated by the LFSR. Only odd numbers will be saved for further processing.

B. Miller Rabin Primality Check

Primality Tester is implemented using Miller Rabin Primality Test which is simulated in Xilinx ISE.

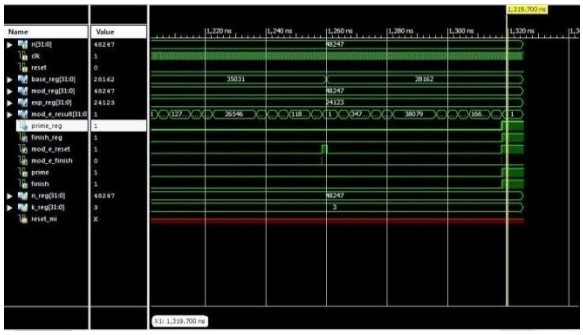

Figure.5 Simulated Waveform for Primality Tester

Figure 5 shows the simulation of Primality test, the input is 2750263. The tester checks whether the number is prime or not and after processing returns that result 2750263 as prime by changing the value of prime_reg to 1.

C. Extended Euclidean

Figure 6 shows the Extended Euclidean algorithm simulated on Xilinx ISE 13.2.

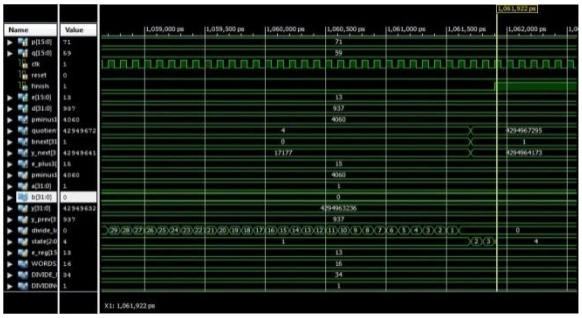

Figure 6: Simulated Waveform for GCD

The waveform for extended Euclidean algorithm for the calculation of GCD and modular inverse shows that two inputs are given p=71 and q=59, as a result the algorithm gives e=13 and d=937 and output.

D. Modular Exponentiation

Modular exponentiation is used for encryption and decryption. There simulated waveforms are shown in Figure 7 and Figure 8 respectively.

1) Encryption

Figure 7 shows the encryption of a plain text, n is set to 4189 and the exponent is 13 i.e. encryption key. The plain text is 101.

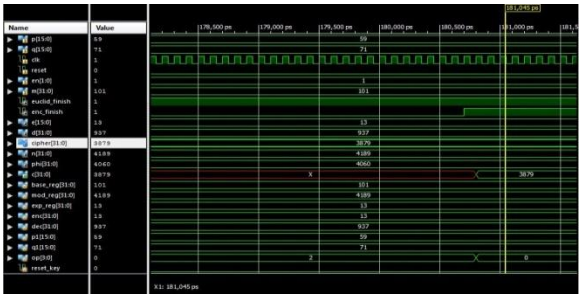

Figure7: Simulated Waveform for Encryption

After encryption the resulted cipher text is 3879.

2) Decryption:

Figure 7 and 8 show the decryption of the cipher text. The value of n remains the same, but for decryption the exponent's value changes to 937 i.e. the value of decryption key. The cipher text is 3879.

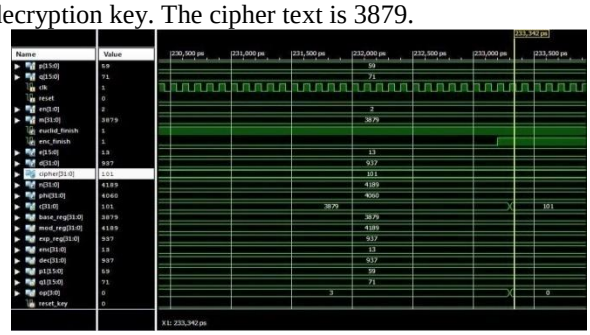

Figure 8: Simulated Waveform for Decryption

After decryption it converts the cipher text back to the original message i.e. 101.

Table 3: Hardware performances

Algorithm Name	1024-bit RSA Encrypt/ decrypt	160-bit MQQ Encrypt/ decrypt	128-bit AES Encrypt/ decrypt	64- bits RSA Encrypt/ decrypt
FPGA type	Virtex-5 XC5VL X30-3	Virtex-5 XC5VL X70T-2	Virtex-5	XILINX 13.2 Verilog
Frequency	251MHz	276.7/24 9.7 MHz	325MHz	254MHz
Throughput	40Kbps	4427Gb ps/399.0 4Mbps	3.78 Gbps	9.8Mbps

We have compared our results with other implementations using throughputs rates as shown in table 3. The results of our implementations show that in hardware, the proposed method for public key algorithm in encryption and decryption is faster when compared with other algorithms.

VII. CONCLUSION

In this paper we have proposed an efficient Primality testing algorithm which helps in generating the random prime number for the key generation procedure, as the key generation is the most essential part of RSA algorithm. Also we implemented a 64-bit RSA circuit in Verilog. It is a full-featured and efficient RSA circuit which includes Primality testing, key generation, data encryption and data decryption. We have implemented random number generator using 32-bit LFSR, Miller-Rabin Primality test, GCD and modular inverse algorithm using extended Euclidean algorithm and Encryption and Decryption using Modular multiplication and modular exponentiation algorithms (R-L binary algorithms). Each sub-component and top module of RSA was simulated in Xilinx and proved functionally correct. This can easily scale up to large bits such as 512 or 1024 or even longer.

ACKNOWLEDGMENT

The authors would like to express sincere gratitude to the Research Centre, Sir Syed University of Engineering and Technology for providing fund for the research.

REFERENCES

[1] Symeon (Simos) Xenitellis, “A guide to PKIs and Open–source Implementations”, The Open–source PKI Book

[2] VibhorGarg, V. Arunachalam ,“Architectural Analysis of RSA Cryptosystem on FPGA”, International Journal of Computer Applications (0975 – 8887) Volume 26– No.8, July (2011).

[3] William Stallings, “Cryptography and Network Security Principals and Practices”, 4th edition, Pearson Education, Inc., (2006).

[4] Sushanta Kumar Sahu, ManoranjanPradhan, “FPGA Implementation of RSA Encryption System”, International Journal of Computer Applications (0975 – 8887), Volume 19– No.9, April (2011)

[5] Monier, L. "Evaluation and Comparison of Two Efficient Probabilistic Primality Testing Algorithms." Theor.Comput.Sci. 12, 97-108, (1980).

[6] Rabin, M. O. "Probabilistic Algorithm for Testing Primality." J. Number Th. 12, 128-138, (1980).

[7] Joe Hurd, “Verification of the Miller-Rabin Probabilistic Primality Test”, Computer Laboratory, University of Cambridge

[8] Chia-Long WU, “An Efficient Montgomery Exponentiation Algorithm for Cryptographic Applications”, INFORMATICA, Vol. 16, (2005) No. 3, 449–468.

[9] AnkitAnand, Pushkar Praveen, “Implementation of RSA Algorithm on FPGA”, International Journal of Engineering Research & Technology (IJERT), Vol. 1 Issue 5, July –(2012).

[10] http://rosettacode.org/wiki/Miller-Rabin_test.

[11] Alkhatib, Mohammad. "On The Design of Projective Binary Edwards Elliptic Curves Over GF (P) Benefiting From Mapping Elliptic Curves Computations to Variable Degree of Parallel Design", International Journal on Computer Science & Engineering/09753397, 20110401.

[12] Thomas Wokinger, “High Speed RSA Implementation of FPGA Platforms”, MS Thesis, Institution of Applied Information Processing and Communications, Graz University of Technology, (2005).

[13] Mohammad El- Hahidy, Danilo G. and Sevin J. K., “High Performance Implementation of public key block cipher-MQQ, for FPGA Platforms”,(2004),eprint.iacr.org/2008/339.pdf.

[14] Ankit A. Pushkar P., “Implementation of RSA Algorithm on FPGA”, International Journal of Engineering Research and Technology, Vol. 1, Issue 5, pp. 1-7, (2012).

[15] Adleman. L. M, C. Pomerance and R. S. Rumely, 1983, “On distinguishing prime number from composite numbers,” Ann of math. (2) 117:1, pp. 173-206.

[16] Artjuhov M. M., “Certain criteria for primality of numbers connected with the Farmat little theorem,” Acta Arith 12, 1966, pp. 55-364.