International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Implementation, Simulation and Synthesis of RSA Cryptosystem

Rafeek Alas[1], Dr. Kiran Bailey[2]

[1]MTech Student, Department of Electronics, BMSCE, Bangalore, India

[2]Assistant Professor, Department of Electronics and Communication, BMSCE, Bangalore, Karnataka, India

Abstract: ***In this paper, we present a methodology to develop 64-bit RSA encryption engine on FPGA 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. The algorithm also requires random prime numbers for processing and generation of public and private key. 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 using Cadence RC Compiler tool and simulated in ModelSim-Altera Student Edition.***

Keywords: Cadence RC, FPGA, ModelSim-Altera, Private Key, Public Key, RSA, Synthesize**.**

1. Introduction

Means and amount of communicated information has changed a lot since last two or three decades. Specially, the amount of information communicated electronically has grown and is growing exponentially fast. It is very much important to develop new ways to guarantee security over the communication channels. So, in order to deal with this large amount of data, a high performance encryption system is needed which processes the data and provides security to the overall electronic information system.

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 usage of communication system increases so does the need for securing data over those channels. Many algorithms are designed to meet these needs.

Cryptographic algorithms, being the core component of most security systems, are usually based upon the fact that their complexity is superior to present computing power. According to the Moore's Law computing power doubles every 1.5 years [1], the complexity of cryptographic computations needs to grow at least at the same rate to provide a consistent level of security. But this does also mean that the actual workload of data processing, meaning encryption and decryption, increases. As a consequence, the demanded amount of computing power of a secured information system increases at the same speed as cryptographic complexity and integration level of its hardware components [2]. One of the algorithms which the above mentioned problems is RSA which is the most widely used public key algorithm. A vast numbers and wide varieties of works have been done on this particular field of hardware implementation of RSA encryption algorithm. Today, RSA is used in IP data security (IPSEC/IKE), transport data security (TLS/SSL), email security (PGP), terminal connection security (SSH), conferencing service security (SILC) and so on.

The security of RSA revolves around the difficulty of factoring a number into two prime factors. Since RSA encryption and finding prime numbers are both computationally intensive tasks, we thought it would be interesting to implement them in hardware on the FPGA to see how these algorithms can be implemented in an application specific integrated circuit. In addition, many companies such as Oracle and Intel have added on-chip hardware support for encryption such as AES to their products. This paper may help us understand the complexities involved in such implementations.

1.1 Cryptography

Cryptography, thus, literally means the art of secret writing. The art of hiding information therefore is not as modern as one might guess but is known to be some thousand years old. Cryptography provides, amongst others, means of hiding and recovering information called encryption schemes. In general an encryption scheme consists of a set of encryption and decryption operations each associated with a key, which is supposed to be kept secret. There are two main categories of encryption: symmetric or public key (asymmetric) encryption mentioned as follows:

1.1.1 Symmetric Encryption

An encryption scheme is related to symmetric cryptography, when it is computationally easy to discover the second key, knowing one of them. In most practical cases the two keys will be identical, which is illustrated by the word symmetric, the shared key is referred to as secret key [3]. A disadvantage is that all parties involved in the communication process have to share a common secret, the secret key. This implicates more difficulties, than might be obvious at first sight. "A common image to explain the idea of symmetric encryption is a safe. All participating parties own an identic copy of the key to the safe, so every party can open the safe to either put something inside (encryption), or to get something out (decryption)."[2]

1.1.2 Public Key (Asymmetric) Encryption
An encryption scheme is said to be public key encryption, when it is impossible to compute the second key, knowing one of them. In this context the encryption operation, using the encryption key, can be regarded as a trapdoor one way function, with the decryption key being the trapdoor that allows easy message recovery. Message recovery without knowledge of the decryption key is computationally infeasible [3].

A major advantage of a scheme belonging to this category is the fact, that one cannot compute the decryption key knowing only the encryption key. This allows distribution of a party's encryption key over insecure channels, which simplifies the process of key distribution. Therefore the encryption key is referred to as public key while the decryption key is called private key.

One of the disadvantages of public key encryption is its bad performance in terms of throughput. In order to keep the decryption key secure, even though the encryption key is available in public, the encryption scheme needs to be more complex than a symmetric one. This denotes that the operations being performed become more complex and time consuming. "To get an idea of Public Key Encryption, one can imagine a simple mailbox. Anyone can put a letter into the mailbox (public encryption key), but only the owner of the mailbox's key can get the letters out of it (private decryption key)."[2]

2. RSA Public Key Encryption

The problems with private key encryption would be resolved if the decoding mechanism could not be (easily) obtained from the encoding mechanism. But how do you get something like that?. The answer is to make, breaking the codes depend on being able to solve a hard problem, like the factorization of a large number.

The RSA cryptosystem is by far the most used public key encryption system. Its name is an abbreviation of the names of R.Rivest, A. Shamir and L. Adleman, who published it in 1978 [5].

This is the most commonly used public-key cryptographic algorithm, and it is considered secure when sufficiently long keys are used. The security of RSA depends on the difficulty of factoring large integers [6].

This section provides a short introduction to the underlying mathematical principles, a detailed look at RSA encryption and decryption operations.

2.1 Mathematics behind RSA

2.1.1 Prime Numbers
An integer **p** larger than 1 is called a prime number if its only divisors are 1 and p, **e.g**. **p** = 2, 3, 5, 7, 11, 13... . There exists an infinite set of prime numbers and there are several well-known algorithms of generating prime numbers. Two integers a and b are called relatively prime, if their greatest common divisor is 1, **e.g**. 3 and 4 are relatively prime. Prime numbers play an important role in public key encryption as will be seen later on.

2.1.2 Modular Arithmetic's
Although most people would say they do not know modular arithmetic or modular reduction they use it in everyday life [2]. Modular reduction means that the set of integer numbers available is limited, the limit is set by the so called modulus, denoted by n. Modular arithmetic with the modulus being 5 means, that the set of available numbers consists of { 0, 1, 2, 3, 4 }. Any number bigger than or equal to the modulus has to be reduced by the modulus by subtraction until it equals a number within the set of available numbers; this operation is called modular reduction. "Modular addition is defined by an ordinary addition followed by a modular reduction operation in order to keep the result within the set of available numbers. Let n = 5, a = 3 and b = 2 then $a + b \equiv 0 \pmod n$ since a + b = 5 and $5 \equiv 0 \pmod n$. People often use modular reduction when talking about time as 21:00 is referred to as 9:00, which is nothing else than $21 \equiv 9 \pmod{12}$ "[2].

Modular multiplication, exponentiation and inversion are the most used and important operations. Modular multiplication works exactly the same way as addition: it is an ordinary multiplication followed by a modular reduction operation. $a \times b \equiv 1 \pmod 5$ since $a \times b = 6$ and $6 \equiv 1 \pmod 5$. Modular exponentiation works slightly different; it can be computed as a series of multiplications followed by a modular reduction operation. $a^b \pmod n = a \times a \times a \ldots.. \pmod n$. "In practice the modular reduction operation will be performed after each multiplication to keep the intermediate results as small as possible in order to save memory and to avoid unnecessary big inputs to the next multiplication" [2]. $a^b \pmod n \equiv (((a \times a) \pmod n) \times a) \pmod n \ldots.$. The multiplicative inverse of a (mod n) is a number within the set of available integers satisfying $a \times b \equiv 1 \pmod n$. If b exists, it is unique and denoted by $b = a^{-1} \pmod n$. b exists, if a and n are relatively prime. In the example, a is invertible and the multiplicative inverse of a modulo n is b, as $3 \times 2 = 6 \equiv 1 \pmod 5$. The well-known Extended Euclidean Algorithm can be used to compute the greatest common divisor of a and n. If it is 1, the algorithm computes the multiplicative inverse of a at the same time.

2.2 RSA Encryption/Decryption Algorithm

In order to set up an RSA encryption scheme, several numbers have to be either randomly chosen or computed. Every party that wants to participate in RSA secured communication has to set up an own scheme based on following:

Following are the steps involved in the RSA algorithm:

2.2.1 Key generation
Key generation is the most important aspect of RSA Algorithm.
The steps are as follows:
1. "Select two random prime numbers p and q
2. Calculate n = p x q
3. Calculate ø(n) = (p – 1) x (q – 1)

4. Select integer e such that gcd (ø(n),e) = 1;1<e<ø(n); where e & ø(n) are relatively prime
5. Calculate d = e-1 mod ø(n).

2.2.2 Encryption

Following is the RSA public key encryption - key generation algorithm. "In order to encrypt a message m for Alice, Bob" should follow these steps [2].

- Obtain Alice's authentic public key (n,e).
- Represent the message as an integer m in the interval [0, n-1].
- Compute $c = m^e \pmod n$.
- Send the cipher-text c to A.

2.2.3 Decryption

If Alice wants to read the received message, she should decrypt the cipher- text according to these steps [2].Use the private key d to recover $m = c^d \pmod n$.

2.3 Example

- I pick p = 17,q = 11. n = 17 * 11 = 187.
- I pick e = 3; d = 107, (ed = 321 = 2 ×16 ×10 + 1).
- I post 187; 3.
- You encode the letter J as 10, and put M = 10; then $C \equiv M^e \equiv 10^3 \equiv 1000 \equiv 65 \pmod{187}$ and so you send me 65.
- I compute $C^d \pmod n$, and find $C^d \equiv 65107 \equiv 10 \pmod{187}$.

3. Modular Exponentiation Operation

Modular Exponentiation operation is simplified using square and multiply algorithm. It is done by using right-to-left-binary method. The purpose of using the binary method is to calculate M^e by using the binary expression of exponent e. In binary method the exponentiation operation is broken in to a series of squaring and multiplication. This method is also very useful for speeding 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 it calculates the exponent e and proceeding 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 ek-1=1 then C=M else C=1
For i=0 to k-1
C=C×C
If ei=1 then C=C×M

This algorithm works on the principle 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 .

4. RSA Algorithm Implementation

RSA algorithm is divided into blocks and each block is then implemented.
The Blocks are
1. Prime Number Generation Block(module).
2. Key Generation (Public key and Private key)
3.Encryption
4.Decryption

4.1. Prime Number Generation(Algorithm)

```
Output 1; //first prime number
Output2;//Second Prime number
For i=3 to MAX, in steps of 2,do //i = 3,5,7 etc
found= 0;
for j=2 to i-2 do
if ((i%j) ==0) then //if i is divisible by j
found = 1;
break; //break out of enclosing for loop
end if;
end for;
if(!found) then
output I; //output i as next prime number
end if;
end for;
```

4.2.1 Key Generation (GCD and Extended Euclidian)

For the generation of keys, two prime numbers are extracted from the Prime FIFO. Applying respective operations on these two primes gives n and Φ(n). After Φ(n), a number e is selected which obeys the condition GCD (Φ(n),e) = 1, means that e is relatively prime to Φ(n). This will prove that modulo inverse of e exists. e × d mod(Φ(n)) = 1

Extended Euclidian algorithm is used and implemented for this purpose. When the GCD is 1, the module returns the values of e and the modular multiplicative inverse d. Otherwise, e gets an increment of 2 and GCD is calculated again, this is repeated 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 Block diagram in Fig 3.2 and Pseudo code for Extended Euclidian algorithm is in Fig 3.3

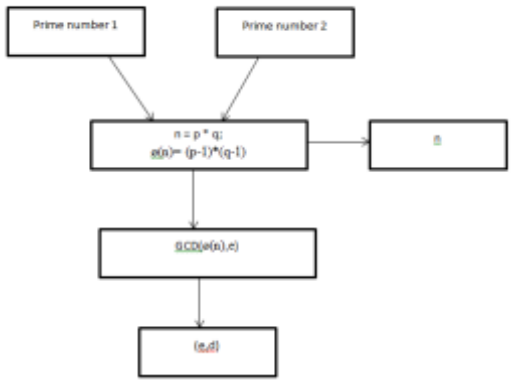


Figure 3.2: Key Generation (e – public key. d –private key)

4.2.2. Pseudo Code of Extended Euclidian

```
Extended_ecuclidian_main(p,q)
e :=1
(gcd,d) := (0,0)
While (gcd :=1 || d < 0)
Begin
e := e + 2
(gcd,d) :=Extended_euclidian_loop((p-1)(q-1),e)
end
```

```
return (e,d)
extended_euclidian_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(p-1)(q-1),e) is a , inverse is y_prev
```

4.3 Encryption/Decryption

Encryption is the process of converting plain text in such a way that eavesdroppers or hackers cannot read it, it is called as cipher text. Decryption is the inverse process by which cipher text is converted back into the form that is readable namely plain text. After generation of the keys, 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. It involves few modular operations like modular addition, modular subtraction and modular multiplication.

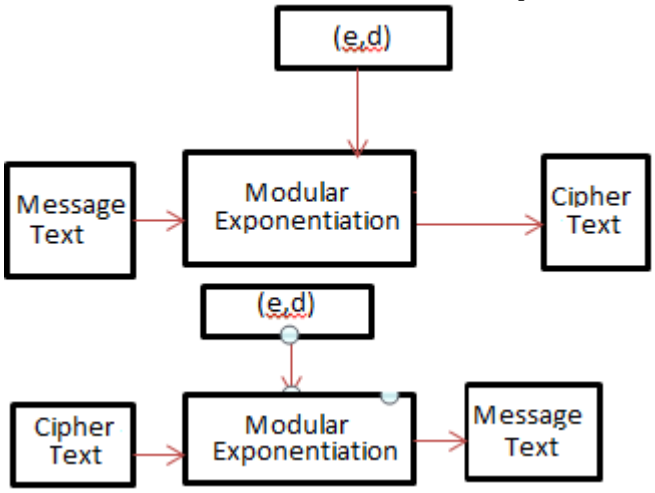


Figure 3.4: Encryption and Decryption

5. Hardware Implementation

For hardware design and implementation, the RSA Cryptosystem is divided into 4 modules.
i. Initial module
ii. Modular exponentiation
iii. Core algorithm
iv. Top module

5.1 Initial Module

This module consists of a Prime number generation module for generating prime numbers for the algorithm, Then these numbers are used for public key and private key generation. The exponentiation part of this algorithm is done by using the right to left binary algorithm implemented for encryption/decryption.

5.2 Modular Exponentiation

The most important and time consuming part of RSA algorithm is calculating the modular exponentiation of a number. 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. The exponent function is also required in miller and Rabin tester so this module can be called there for processing and calculating, saving both space and time.

5.3 Core Algorithm

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

When a new used comes to the system this module takes two numbers as input. n and Φ(n) are calculated by inserting them into the multiplier, Φ(n) is then used to find the encryption and decryption keys. For this purpose the Euclidean Algorithm is used which calculates the GCD of Φ(n) and an odd number. If the GCD is found to be 1 this proves that the modular inverse of the odd number exists and the number is co-prime. This number is then saved in a register and will be used as the encryption key. The Extended Euclidean Algorithm is then used to find the decryption key which is the modular inverse of the encryption key. Hence the key generation process is completed for this user and he is allotted with a public and private key that user will use to encrypt and decrypt the data. Once the keys are created they can be used for encrypting and decrypting the message. The Modular Exponentiation is used for this purpose.

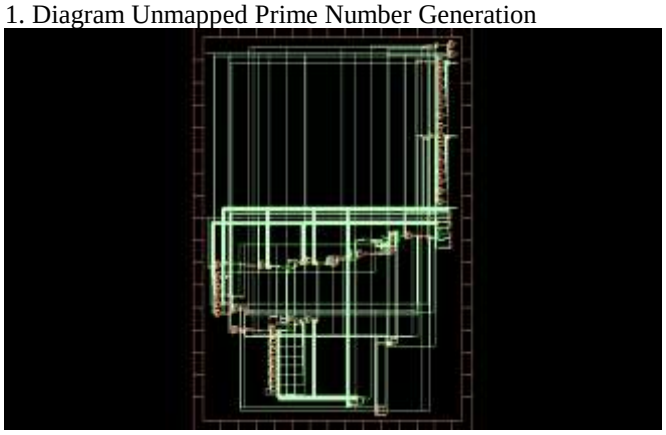
5.4 Top Module

The top module controls the functions of the other modules and interconnects them so as the RSA Algorithms flow is maintained. This module implements a controller with multiple checks so as to get the desired results.

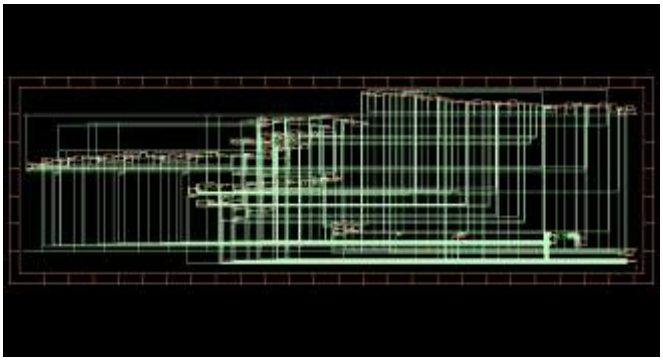
Synthesis of RSA Cryptosystem

6.1 Synthesis is carried out in Cadence RC compiler

1. Diagram Unmapped Prime Number Generation



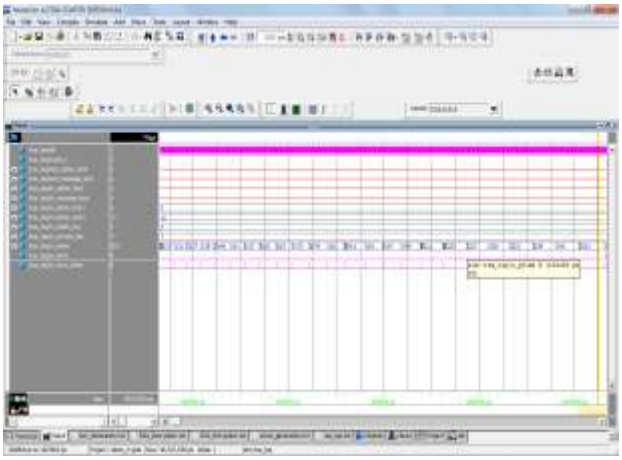
2. Diagram for Mapped Synthesis Prime Number Generation.



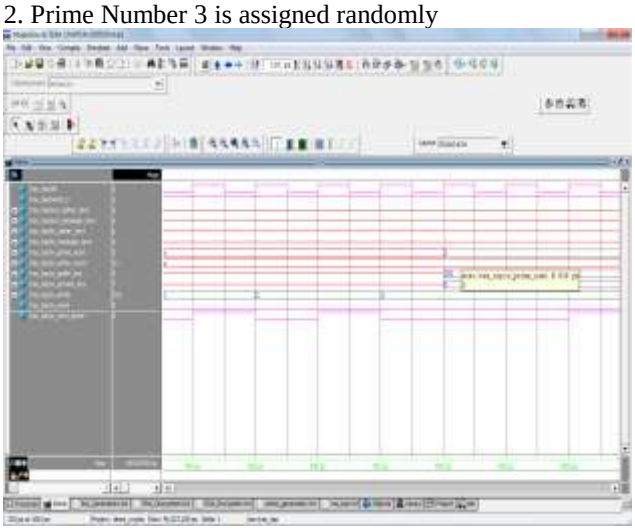
6. Simulation Results

Prime number generation module Extended Euclidean and modular exponentiation have been successfully written and tested on Xilinx ISE 13.2 and Modelsim. The simulation shows the desired results of these algorithms. The following section shows the simulation results of these algorithms.

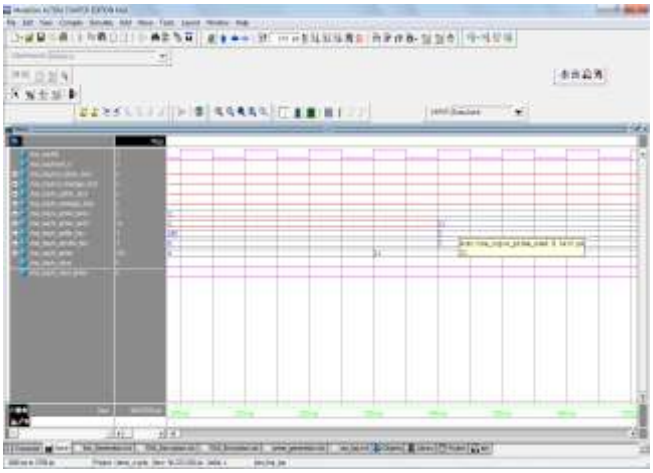
1. Prime Number generation till 255 in figure prime numbers are shown form 127 to 251.



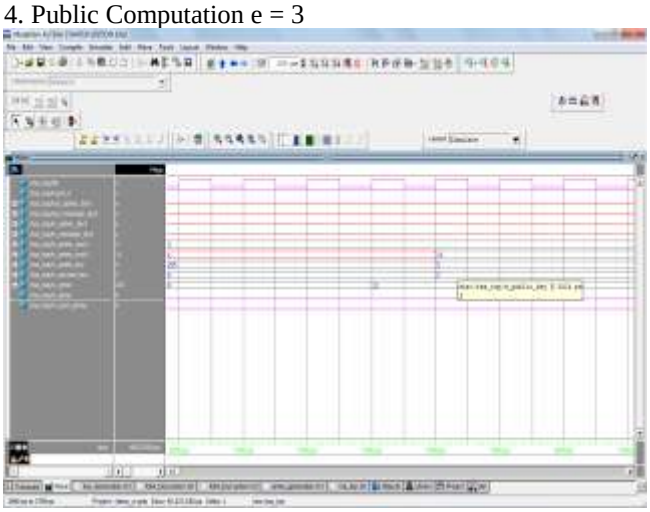
2. Prime Number 3 is assigned randomly



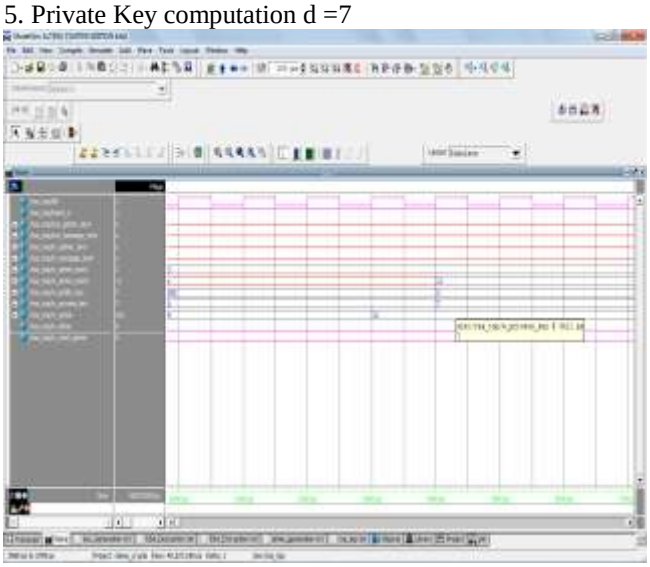
3. Prime number 11 is assigned randomly



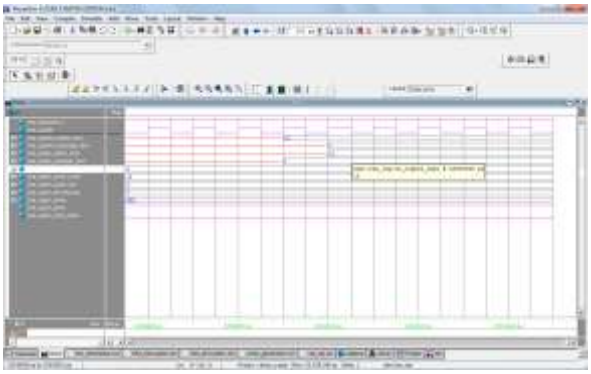
4. Public Computation e = 3



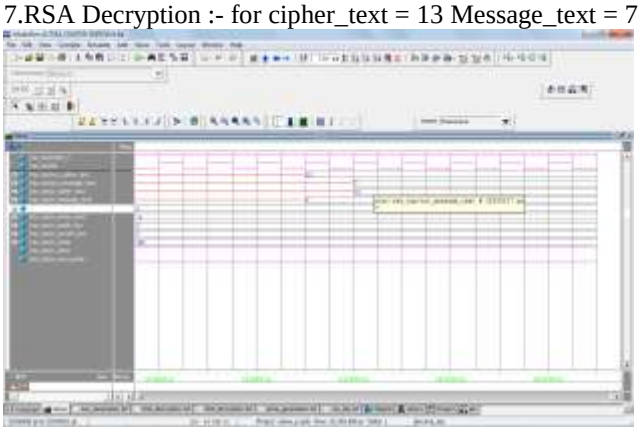
5. Private Key computation d =7



6. RSA Encryption: - Message_text = 7 cipher_text = 13



7.RSA Decryption :- for cipher_text = 13 Message_text = 7



7. Conclusion

Here we implemented a 64-bit RSA circuit in Verilog. It is a full-featured and efficient RSA circuit this includes Prime generation, key generation, data encryption and data decryption. We have implemented random prime number generator using Prime number generation module, 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, Modelsim and proved functionally correct. This can easily scale up to large bits such as 512 or 1024 or even longer.

8. Acknowledgment

This research is supported by the BMS College of Engineering, Bangalore. The authors wish to thank BMS college of Engineering for supporting this work by encouraging and supplying the necessary tools.

References

[1] Moore, Gordon E. (1965): Cramming more components onto integrated circuits Electronics, Volume 38, Number 8
[2] Benedikt Gierlichs: Hardware-Software Co-Design: A Case Study on an accelerated Implementation of RSA
[3] Menezes, van Oorshot, Vanstone (1997) Handbook of applied cryptography, CRC Press
[4] M.K. Hani, T.S. Lin, N. Shaikh-Husin: FPGA Implementation of RSA Public-Key Cryptographic Coprocessor Proceedings of TENCON, vol. 3, pp. 6-11, Kuala Lumpur, Malaysia, 2000
[5] R.L. Rivest, A. Shamir, and L. Adleman (1978): A Method for Obtaining Digital Signatures and Public-Key Cryptosystems Communications of the ACM 21,2, pp. 120-126
[6] Ankit Anand, Pushkar Praveen: Implementation of RSA Algorithm on FPGA Centre for Development of Advanced Computing, (CDAC) Noida, India

Author Profile

Rafeek Alas received B.E degree from S D M College of Engineering and Technology, Dharwad, Karnataka, India in 2011. he is currently pursuing her MTech degree in the field of Electronics, department of Electronics and Communication in BMS College of Engineering, Bangalore, India. Her fields of interest in research are VLSI design and VLSI circuits..

Dr. Kiran Bailey received her B.E degree from Dayananda College of Engineering of Bangalore University, Bangalore in the year 1997. She got her M.Tech degree from B.M.S. College of Engineering of Visvesvaraya Technological University, Bangalore in 2001. She joined BMSCE in 1998 and has since been teaching Electronics related subjects. Her areas of interest are solid state devices, VLSI design, Low power VLSI circuits. Presently she is an Associate professor in the dept. of E&C, BMSCE, Bangalore.