



International Journal of Engineering, Business and Enterprise Applications (IJEBEA)

www.iasir.net

EFFICIENT IMPLEMENTATION OF RSA ON FPGA USING VERILOG

A.Santham Bharathy ME (VLSI)

Department of Electronics & Communication Engineering,
Anna University-Chennai/ Nehru Institute of Technology,
Jawagar Garden, Kaliyapuram, Coimbatore, Tamil Nadu, INDIA

Abstract: This research paper is aimed to implement the RSA algorithm 1024-bit in the FPGA with the help of Verilog HDL. The RSA algorithm using FPGA can be used as a standard device in the secured communication system. A simple nested loop addition and subtraction have been used in order to implement the RSA operation. The modification of RSA algorithm includes pipeline and data dependence computational block. This results in very low frequency, high speed, low power consumption and low cost compared to other algorithm methods. The beneficial of a pipelining approach is provide a clock input to each modules and enabling them each time the output from the previous module is available. The data dependence computational block is used to provide low power dissipation by reducing unnecessary switching takes place in the architecture. The information to RSA encryption side is in the form of statement and the same will appear in the decryption side and its real time input/output also achieved effectively. Each sub-component and module of RSA was simulated in Xilinx Spartan 3E software tools and provide functionally correct.

Keywords: Cryptography, FPGA, Verilog, Security, Communication.

I. Introduction

Public-key cryptography uses asymmetric key algorithms (such as RSA), and can also be referred to by the more generic term "asymmetric key cryptography." The algorithms used for public key cryptography are based on mathematical relationships that presumably have no efficient solution. Although it is computationally easy for the intended recipient to generate the public and private keys, to decrypt the message using the private key, and easy for the sender to encrypt the message using the public key, it is extremely difficult (or effectively impossible) for anyone to derive the private key, based only on their knowledge of the public key. Advantages of asymmetric key algorithm are It solves the problem of distributing the key for encryption. Everyone publishes their public keys and private keys are kept secret.

Public key encryption allows the use of digital signatures which enables the recipient of a message to verify that the message is truly from a particular sender. The use of digital signatures in public key encryption allows the receiver to detect if the message was altered in transit. A digitally signed message cannot be modified without invalidating the signature. This is why, unlike symmetric key algorithms, a public key algorithm does not require a secure initial exchange of one (or more) secret keys between the sender and receiver. The use of these algorithms also allows the authenticity of a message to be checked by creating a digital signature of the message using the private key, which can then be verified by using the public key. In practice, only a hash of the message is typically encrypted for signature verification purposes.

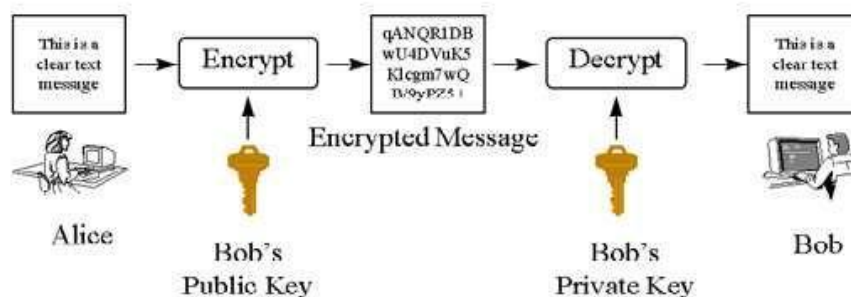


Fig. 1: Asymmetric-key cryptography

This work approaches hardware implementation of RSA algorithm scheme using the modular exponentiation operation. Simple nested loop addition and subtraction have been used to implement the modular exponentiation

operation. And to implement this, only shift registers, XORs and LUTs are used. The usage of NAND gate is avoided to reduce the complexity of the circuit by employing with reusability characteristics of XORs. Here, it supports multiple key sizes for RSA according to the application requirement. This new approach helps to reduce the system processing time, gate counts, frequency requirement and power consumption. And also the system could take the information in the form of statement say word format (real time input/output) not in binary or hex format as the earlier approaches handled.

II. Design Overview

A distinct feature that can be found in the RSA algorithm is that it concedes most of the integral part used in encryption to be re-used in the decryption process, which can decrease the resulting hardware area. In RSA, a plaintext block M is encrypted to a cipher text block C by:

$$C = M^e \bmod n \quad (1)$$

The plaintext block is recovered by:

$$M = C^d \bmod n \quad (2)$$

RSA encryption and decryption are mutual inverses and commutative as shown in equation (1) and (2), due to symmetry in modular arithmetic. One of the potential applications for which this design of RSA has been targeted is the secured data communication. In this application, the data input could be a statement which is fed into FPGA board directly via serial communication. The encryption module takes care of the security. The process at the receiving end is same as the process that has been followed at the sending end except that the sequence of the module is reverse. The RSA covers both the operation of encryption and decryption.

A. Fundamental RSA process

The RSA algorithm needs estimation of the modular exponentiation, which is separated into a series of modular multiplications by the application of exponentiation inquiring. The RSA encryption process is the mathematical operation, $c = m^e \bmod n$. This mathematical operation has involved a few modular operations like modular-exponentiation, multiplication, addition and subtraction process on large integers. Detail algorithms of the above operations for hardware implementation have been discussed in the following sections

B. Modulus Exponentiation Process

The modular exponentiation operation is simply an exponentiation operation where multiplication and squaring operations are modular. The exponentiation operation developed for computing M^e are applicable for computing $M^e \bmod n$. In the concern of hardware implementation, a clever algorithm is required in order to extent a superior efficiency. Hence, exponentiation is acquired by doing a number of squaring and multiplications.

C. Modular Multiplication Process

The modular multiplication problem is defined as the computation of $P = (A \times B) \bmod n$, given the integers A , B , and n . It is usually assumed that A and B are positive integers with $0 \leq A, B < n$. The modulus multiplication operation is required after the separation of exponentiation into a number of squaring and multiplication. There are basically four general approaches for computing the product. Multiply and then divide, Interleaving multiplication and reduction, Brickell's method and Montgomery's method. All the above approaches have a common disadvantage that it doubles up the number of bits for each multiplication. For example, when two 32-bit numbers are multiplied together will cost a 64-bit result and hence a large register is needed to store this result. A modified algorithm is used in this design which will be discussed later. The modified algorithm overcomes the problem by separating the multiplication operation into a number of modular addition operations.

D. Complete Algorithm

1. Generate two large random primes, p and q , of approximately equal size such that their product $n = p \times q$ is of the required bit length, e.g. 1024 bits.
 2. Compute $n = p \times q$ and $\phi = (p-1)(q-1)$.
 3. Choose an integer e , $1 < e < \phi$, such that $\gcd(e, \phi) = 1$.
 4. Compute the secret exponent d , $1 < d < \phi$, such that $ed \equiv 1 \pmod{\phi}$.
 5. The public key is (n, e) and the private key (d, p, q) . Keep all the values d, p, q and secret. [We prefer sometimes to write the private key as (n, d) because you need the value of n when using d .]
- n is known as the modulus.
 - e is known as the public exponent or encryption exponent or just the exponent.

- d is known as the secret exponent or decryption exponent
- Choose $p = 7$ and $q = 17$
- Compute $n = p * q = 7 * 17 = 119$
- Compute $\phi(n) = (p - 1) * (q - 1) = 6 * 16 = 96$
- Choose e such that $1 < e < \phi(n)$ and e and n are coprime. Let $e = 5$
- Compute a value for d such that $(d * e) \% \phi(n) = 1$. One solution is $d = 77$ [$77 * 5 = 385 = 4 * 96 + 14$]
- Public key is $(e, n) \Rightarrow (5, 119)$
- Private key is $(d, n) \Rightarrow (77, 119)$
- The encryption of $m = 19$ is $c = 19^5 \% 119 = 2476099 \bmod 119 = 66$
- The decryption of $c = 66$ is $m = 66^{77} \% 119 = 19$

III. Modification

A. Pipelining

The encryption and decryption portion in the core consists of many 'FOR' loops which are again synthesized as a series of identical modules connected in series. Whenever new data appears at the input each module does a small portion of the encryption operation and passes it to the next module. Each module remains idle after before and after performing its operations. This path is connecting all the modules is also the critical path which decides the clock period and hence the throughput of the system. It is hence beneficial to follow a pipelining approach by providing a clock input to each of these modules and enabling them each time the output from the previous module is available. This increases throughput, so that output appears at the end of each clock cycle. The pipelined architecture is shown below.

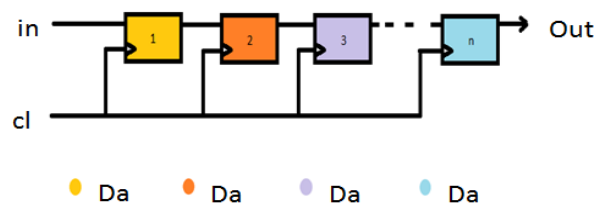


Fig. 2: Pipelining

B. Data Dependent Computation Block

The architecture shown in Fig is used for intra data dependencies analysis. The data correlation between the next input is analyzed and based on the dependencies certain arithmetic operation can be skipped up so as to reduce the computation and reduce the critical path of the computation. The correlation finder block detects the correlation and pass appropriate control signal to enable and disable the multiplier and adder. If the detector comes across the subsequent

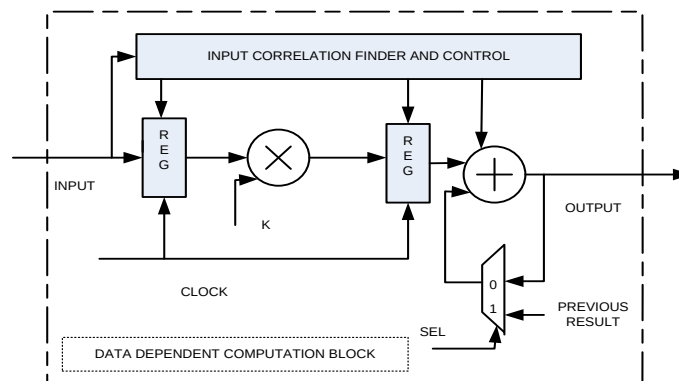


Fig. 3: Data Dependent Computation Block

IV. VHDL MODELING

A. Device Utilization Summary

From device utilization summary, it is clear that total equivalent gates used for the approach is 951 only. And apart from the number of slices, all other components utilized percentile is less than 10 which includes XORs, LUTs and shift registers. Because of total equivalent gate count is very low that results in very low frequency requirement to

perform this operation, low power consumption and low cost compared to earlier methods.

Logic utilities	Available	Used	% of Use
Device name	XC3S400		
Number of Slices	4656	518	15
Number of Slice Flip Flops	7168	459	13
Number of 4 input LUTs	9312	936	14
Number of IOS	232		
Number of bonded IOBS	232	83	33
Number Of GCLKS	121	21	10

V. SIMULATION, SYNTHESIS AND DISCUSSION

VHDL is commonly used to write text models that describe a logic circuit. Such a model is processed by a synthesis program, only if it is part of the logic design. A simulation program is used to test the logic design using simulation models to represent the logic circuits that interface to the design. This collection of simulation models is commonly called a test bench. VHDL has file input and output capabilities, and can be used as a general-purpose language for text processing, but files are more commonly used by a simulation test bench for stimulus or verification of data. There are some VHDL compilers which build executable binaries. Here, VHDL program is used for RSA, data control, UART and device drivers to write a test bench, to verify the functionality of the design using files on the host computer to define stimuli, to interact with the user, and to compare results with those expected. After the generation of codes that simulates successfully, it may not be synthesized into a real device or is too large to be practical. Once synthesis is over, the input message is in the form of data can be given as input to FPGA from computer via into a real device or is too large to be practical Hyper Terminal or Ethernet in serial communication media. This message is moved to FPGA in binary form with selected bit values one by one. Now, RSA algorithm which is implemented using VHDL program in FPGA processes this message and produces encrypted data as output in the hyper terminal screen. So, the output, which is not in readable format, can be saved in note pad. Now, the encrypted file can be sent for the decryption process to get the original message. All these operations have been carried out in the Spartan 3E FPGA electronic module and its real time output could be seen through hyper terminal screen and not in the form of binary or hex formats. Hence, this FPGA module can be used as a standard device in the secured closed network data communication system.

So, a step has been taken to design hardware in a VHDL IDE for FPGA implementation using Xilinx ISE to produce the RTL schematic of the desired circuit. After that, the generated schematic can be verified using simulation software which shows the waveforms of inputs and outputs of the circuit after generating the appropriate test bench. Finally, the VHDL model is translated into the gates and wires that are mapped onto a programmable logic device FPGA. Hence it is the actual hardware which is configured as processor chip rather than the VHDL code which is used for implementing the RSA Algorithm.

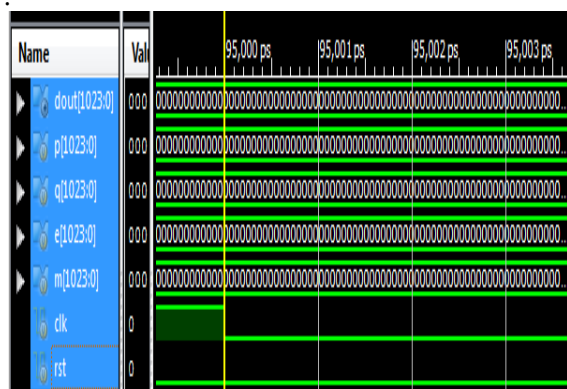


Fig. 4: Output Waveform for 1024 Encryption

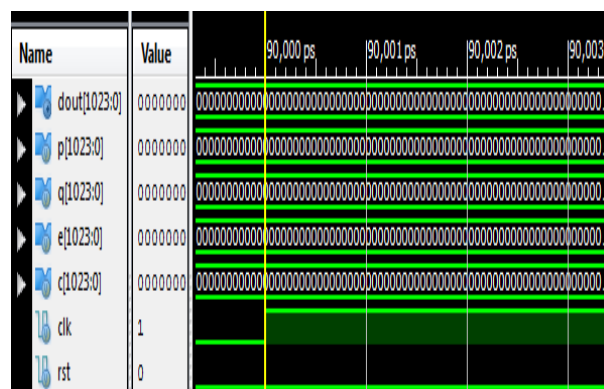


Fig. 5: Output Waveform for 1024 Decryption

VI. CONCLUSION

The detailed implementation techniques for 1024-bit RSA encryption and decryption algorithm were presented. The modular exponential for encryption and decryption process is performed by using pipeline and data computational block components in algorithm. The add and shift algorithm is used to perform the modular multiplier where the algorithm are implemented using Verilog code targeting Virtex-5 xc3S500e-4fg320 FPGA Xilinx. The whole design is tested and each sub-component of module of RSA are simulated using Xilinx Spartan 3E device and provide functionally correct. The future work that can be done in this regard includes, multiple key generation from 1 to 1024-bit encryption and decryption that enhance more secure for information securing platform.

REFERENCES

- [1]. Adam J.Elbert, W. Yip, B-Chetwynd, and C. Paar, —An FPGA Based Performance Evaluation of the RSA Algorithm IEEE Finalist TRANSACTION VLSI SYSTEMS,VOL.9, NO.4, AGUGEST-2012.
- [2]. Chen-Hsing Wang, Chieh-Lin Chuang and Chang-Wen Wu|An efficient Multiplier Supporting RSA Public-Key Cryptosystems| IEEE TRANSACTION, VOL. 18, NO.4, April-2011.
- [3]. Devendra N. FPGA Kayatanavar,—Implementation of RSA Encryption and Decryption —International Conference on Control, Automation, Communication and Energy Conservation-2009, 4th-6th June 2009.
- [4]. FPGA Implementation of RSA Cryptosystem|, International Journal of Engineering and Applied Sciences 2:3 2010.
- [5]. International Journal of Reconfigurable and Embedded Systems (IJRES), —RSA Encryption Algorithm Hardware Implementation: Throughput and Area Comparison| 4864ISSN:Vol.1,No.20892,July-2012.
- [6]. M. D. Shieh, J. H. Chen, H. S.Wu, and W. C. Lin, "A new modular exponentiation Architecture for efficient design of RSA cryptosystem," IEEE Trans. Very Large Scale Integration. (VLSI) Syst., vol. 16, no. 9, pp. 1151-1161, Sep. 2010.
- [7]. M. D. Shieh, J. H. Chen, H. S.Wu, and W. C. Lin, "A new modular exponentiation Architecture for efficient design of RSA cryptosystem," IEEE Trans. Very Large Scale Integration.(VLSI) Syst., vol. 16, no. 9, pp. 1151-1161, Sep. 2010.
- [8]. S. Nimmagadda, Oforomance.Elkeelany,evaluation of —P different hardware models of RC5 of The IEEE 39th Southeastern Symposium on System Theory, 2010.
- [9]. William Stallings, Cryptography and Network Security Principal and Practices, 4th Edition, Pearson Education, Inc., 2006, ISBN 81-7758-774-9.
- [10]. Xinmiso Zhang , IEEE and Keshab K. Parhi Fellow, High IEEE-Speed VLSI Architectures for the RSA Algorithm| IEEE TRANSACTIONS, VOL.12, NO:9, SEP-2010.
- [11]. W. Stallings, Cryptography and Network Security: Principles and Practice 3/e. Prentice Hall.