Implementation of RSA Algorithm on FPGA

Ankit Anand, Pushkar Praveen
Centre for Development of Advanced Computing, (CDAC) Noida, India

Abstract

This paper presents the design and implementation of a flexible key RSA encryption that can be used as a standard device in the secured communication system based on Montgomery algorithm. The VHDL modelling of this RSA encryption design has the unique characteristics of supporting multiple key sizes, and it can easily be fit into the systems that require different levels of security. In order to implement the RSA operation a simple nested loop addition and subtraction have been used. This has made the processing time faster and used comparatively smaller amount of space in the FPGA. RSA is fully described using VHDL on Xilinx ISE software. Target device 3s1600efg320-4 also belongs to the same company. This is an advantage since Xilinx ISE software provides full support for all the code-to-FPGA processes for Xilinx FPGAs. The RSA encryption implementation has made use of 13,779 units of logic elements and achieved a clock frequency of 69.09MHz.

***Keywords*— Cryptosystem, Encryption, Decryption, RSA, Security, VHDL.**

Introduction

Now these days' electronic data communications and Computer networks have made, it very much important to develop new ways to guarantee their security. As the time passes demand of security in the communication channel increases, and the development of a new and efficient hardware security module has started to get the primary preference. A large number and wide varieties of works have been done on the hardware implementation of RSA encryption algorithm. The hardware implementation of RSA encryption scheme has been proposed by Khalil. Where they use Montgomery algorithm with modular multiplication and systolic array architecture. This design scheme focuses on the implementation of a 1024-bit RSA cryptographic processor. But these design have the drawback of a slower processing time, though some of them use a faster clock. Shand have proposed a software implementation of RSA cryptography. A different approach has been taken by Chris for implementing RSA cryptographic scheme. But, it does not provide the flexibility of using many practical applications as it can only be implemented with a fixed key size.

RSA

The RSA cryptosystem was invented by Rivest, Shamir, and Adleman in 1977. 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. Difficulty of factoring *n to* find the original primes *p,q* defines the strength of RSA. Hence the larger the value of primes, the harder the factorization. Again, typical values for these primes are 512 to 4096bits. We can easily understand it from the algorithm given below

1) Select any prime numbers p, q
2) Compute n = p*q
3) Compute phi = (p-1)*(q-1)
4) Select e, such that 1< e < phi, and gcd(phi, e) = 1
5) Find d such that ed = 1 mod phi
6) Public key KU = {e, n}
7) Private key KR = {d, n}

For any plaintext M < n,

Encryption, $C = M^e \pmod n$

Decryption, $M = C^d \pmod n$

Steps of encryption and decryption transform:

RSA involves a public key and a private key. The public key can be known to everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted using the private key. The keys are generated by following steps:

1. Choose two distinct large random prime numbers p and q.

2. Compute n = pq. The number n is used as the modulus for both the public and private keys.

3. Compute the Euler's function: $\phi(n) = (p-1)(q-1)$.

4. Choose an integer e such that $1 < e < \phi(n)$; e and $\phi(n)$ share no factors other than 1 (i.e. e and $\phi(n)$ are co-primes); e is released as the public key exponent.

5. Compute d to satisfy the congruence relation $de \equiv 1 \pmod{\phi(n)}$; i.e. $ed \equiv 1 \pmod{\varphi(n)}$.

Or, $d = (1 + x\ \varphi(n)) / e$

A popular choice for the public exponents is $e = 2^{16} + 1 = 65537$. Some applications choose smaller values such as e = 3, 5, 17 or 257 instead. This is done to make encryption and signature verification faster on small devices like smart cards but small public exponents can lead to greater security risks. The public key consists of the modulus n and the public (or encryption) exponent e. The private key consists of the modulus n and the private (or decryption) exponent d which must be kept secret and the decryption exponent d has to be greater than $n^{0.292}$, otherwise the RSA crypto- system can be broken.

Mathematics used for RSA

For hardware implementation of RSA, an intelligent algorithm is needed in order to reach a higher efficiency. Hence, exponentiation is achieved by performing a number of squaring and multiplications. Given the integers M, e, and n, the e has to be changed to binary in order to start the algorithm to compute M^e.

Output $C = M^e$ (e contains h-bits)

If $e_{h-1} = 1$, then C: =M else C: =1

For i = h-2 down to 0

a. C: = C x C

b. if e i = 1, then C:= C x M

Return C

Let us assume e = 43 = 1010112. So the h = 6 (e contains 6 bits). Using Left-to-Right method, as $e_5 = 1$, C = M algorithm starts as the following table:

	e_i	A	B
4	0	$(M)^2$	$(M)^2$
3	1	$(M^2)^2$	$(M)^4 * M$
2	0	$(M^5)^2$	$(M)^{10}$
1	1	$(M^{10})^2$	$(M)^{20} * M$
0	1	$(M^{21})^2$	$(M)^{42} * M = M^{43}$

Table 1: LR method of computing exponentiation

Hardware Implementation

During the hardware design the whole process is divided into five modules.

I. Initial module

II. Montgomery multiplication

III. Core module

IV. Final module

V. RSA Top module

(I) Initial Module: This module consist all the inputs, initialize and produce16 bit output. This is then used by the next module as the input.

(II) Montgomery Multiplication: The core performs a classical modular exponentiation. The data needed is the following:

1. Bit size: this is a constant value which specifies the bit length of value y, it is necessary in order to perform private-key exponentiation (The usual value of this

field will be “512”) or public-key exponentiation (It can vary between a few bits). It can be calculated as log2(y) being y the key used to cipher.

2. X: This is the plain text input which will be ciphered.

3. Y: This is the key input, which will be used to cipher X

4. M: This is the module M input.

5. r_c: this is a 512 bit length constant needed by the ciphering algorithm in order to achieve a high performance.

6. Start in: active it when load the first 16 bits of m.

7. valid in: should be active high (logical value of 1 as long as the data is being introduced).

8. S: This port is the data output of the exponentiation.

9. Valid out: as its name says, it indicates when the values on S are valid.

The Montgomery multiplier is the biggest part of the architecture. Depending on the mode signal four different operations are performed. Table shows all available modes of operation.

Mode	Description
00	Pass through the sum and carry
01	Right shift of sum and carry
10	Shift in of interface data
11	Shift out data to the interface

Table 2: Operation mode of the Montgomery multiplier

The pass through mode is used for converting the carry-save representation into the binary representation. The control unit has to ensure that all other inputs except the sum and carry inputs are set to zero. The right shifting of the sum and carry inputs is used during the computation. The third mode is used to load input data. The fourth mode is used the control unit has to ensure that the right inputs were applied at the right time.

The RSA implementation uses an address width of two bits, so four different values can be stored in the single-port RAM. The following values are stored in the single-port RAM (SPRAM):

At address ’00’: r2 is stored which is needed for the transformation of integers from the Z domain to the Montgomery domain.

At address ’01’: the number one is stored which is needed for the transformation from the Montgomery domain to the Z domain.

At address ’10’: is stored and it is replaced with a after the transformation to the Montgomery domain.

At address ’11’: the result of one Montgomery multiplication is stored. It will be overwritten after the next multiplication is finished.

(III) Core Module: This module is used for generating different cores like Single port Ram Core for storing the data, FIFO for first in first out the data and also the FIFO feedback.

(IV) Final Module: This is the last module which computes all the previous output values and produces the 1024 bit data.

(V) Top Module: This module is the control unit for controlling the functioning of the rest of the modules and to ensure that the RSA algorithm flow is followed and maintained. The RSA Core is actually a modular exponentiation and it is realized by mapping the modular exponentiation algorithm onto hardware. The Top modules include the Montgomery Multiplication Block RAM blocks and a Controller. Our design approach is not to hard-wire the modulus into the logic circuit but by storing the operands in RAMs.

Simulated Results:

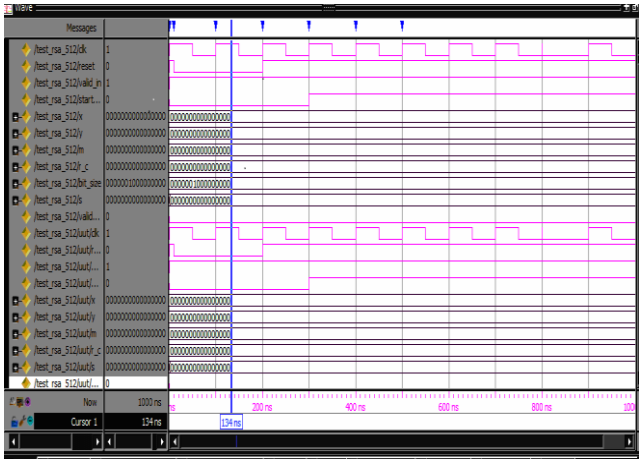

Figure 1: Simulated Result of Top Module

Synthesis Result:

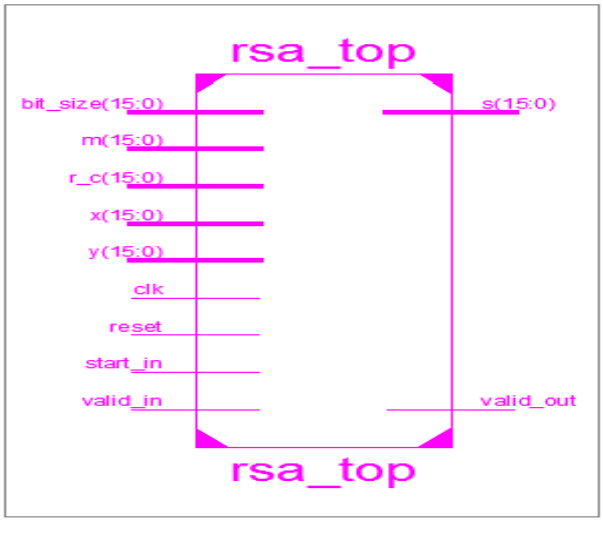

Figure 2: RSA_Top Module Hardware

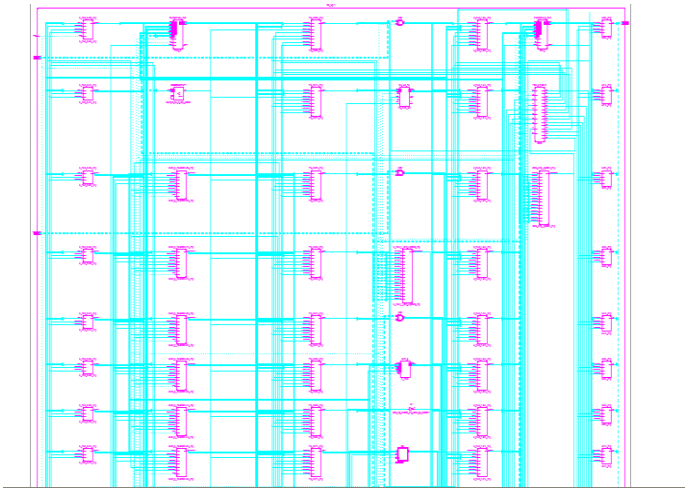

Figure 3: Detailed Schematic of Hardware generation of Top Module of RSA

Device Utilization Summary (estimated values)			[-]
Logic Utilization	Used	Available	Utilization
Number of Slices	8956	4656	192%
Number of Slice Flip Flops	8032	9312	86%
Number of 4 input LUTs	15617	9312	167%
Number of bonded IOBs	95	232	40%
Number of BRAMs	9	20	45%
Number of MULT18X18SIOs	20	20	100%
Number of GCLKs	1	24	4%

Table 3: Device Utilization Summary

Timing Summary:

Minimum period: 14.473ns (Maximum Frequency: 69.093MHz)

Minimum input arrival time before clock: 8.905ns

Maximum output required time after clock: 10.657ns

Maximum combinational path delay: 10.457ns.

Conclusion:

The main aim of this project is the development of high performance RSA. We start with reviewing basic theoretical foundations of RSA. Today's FPGA are ready for RSA implementations operation on full word size, using Montgomery algorithm it is possible to implement RSA with common word sizes like 1024 bits, 1536 bits, and 2048 bits. The results also shows that a proper interface design and a well implemented device driver are needed to provide the same throughput on application layer as on the hardware layer and we have got the waveform successfully after simulation and also got the hardware design for RSA after synthesis.

Refrences:

[1] International Journal of Scientific & Engineering Research Volume 2, Issue 5, May-2011 ISSN 2229-5518.

[2] http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/index.html.

[3] Institute for Applied Information Processing and Communications Graz University of Technology, High-Speed RSA Implementation for FPGA Platforms by Thomas W¨ockinger in January 2005.

[4] Adriana Borodzhieva, Plamen Manoilov, Software Tool for Implementing RSA Algorithm, International Scientific Conference Computer Science'2008.

[5] W. Diffie and M. E. Hellman, .New Directions in Cryptography,. IEEE Trans. on Information Theory, vol. IT-22, pp. 644-654, November 1976.

[6] R. L. Rivest, A. Shamir, and L. Adleman, .A method for obtaining digital signatures and public-key cryptosystems,. Communications of the ACM, vol. 21, pp. 120-126, February 1978.

[7] http://www.di-mgt.com.au/rsa_alg.html.

[8] Peter J. Ashenden. The Designer's Guide to VHDL. Morgan Kaufmann Publishers, 2nd edition, 2002.

[9] Cetin Kaya Koc, "High Speed RSA Implementation", RSA Laboratories, Version 2.0, 1994.

[10] John Fry - Martin Langhammer. "RSA & Public Key Cryptography in FPGA"2000.

[11] C. Mclvor. M. McLoone. I. McCanny. A. IDaly and W. Mamane, "FastMontgomery Modular Multiplication and RSA Crpographic Processor Architectures."37Ih Asilornar Conference on signal, system and computers, nov 2003.

[12] O.Nibouche. A. Bouridane and M. Nibouche, "New Iterative Algorithms and Arclritectures of Modular Multiplication for Ctyptography", Proceedings of the 8"lntemational IEEE Conference on Electronics. Circuits, and Systems, ICECS Malta 2001.

[13] W. L. Freking and K. K. Parhi. Performance-scalable array architectures for modular multiplication. In Proceeings of rhe IEEE lnreniational Conference on Application-Specific Sysrems, Archirecrures, and Processors, pages 149-160. IEEE. 2008.

[14] P. L. Montgomery, "Modular Multiplication without Trial Division," Mathemat. of Computat., vol 44, pp 512-521, April 1985.

[15] Mohamed Khalil, Koay Kah Hoe," VHDL Module Generator: A Rapid-prototyping Design Entry Tool for Digital ASICs," Jurnal Teknologi, 3 1 (D)I 999, Univ. Teknologi Malaysia, Dec. 1999, pp.45-61.

[16] Public-key cryptographic standards. http://www.rsasecurity.com, 2004.

[17] Peter J. Ashenden. The Designer's Guide to VHDL. Morgan Kaufmann Publishers, 2nd edition, 2006.