FPGA Implementation of RSA Encryption System

Sushanta Kumar Sahu
Deptt. of Electronics & Tele-comm. Engg.
VSS University of Technology, Burla.

Manoranjan Pradhan
Deptt. of Electronics & Tele-comm. Engg.
VSS University of Technology, Burla.

ABSTRACT

This paper presents the architecture and modeling of RSA public key encryption/decryption systems. It supports multiple key sizes like 128 bits, 256 bits, 512 bits. Therefore it can easily be fit into the different systems requiring different levels of security. In this paper simple shift and add algorithm is used to implement the blocks. It makes the processing time faster and used comparatively smaller amount of space in the FPGA due to its reusability. Each block is coded with Very High Speed Integrated Circuit Hardware Description Language. The VHDL code is synthesized and simulated using Xilinx-ISE 10.1. It is verified that this architecture support multiple key of 128bits, 256bits, and 512 bits.

General Terms

Security, Algorithms, Cryptography.

Keywords

RSA, VHDL, FPGA, modular multiplication.

1. INTRODUCTION

The art of keeping messages secure is cryptography. Cryptography plays an important role in the security of data. It enables us to store sensitive information or transmit it across insecure networks so that unauthorized persons cannot read it. The urgency for secure exchange of digital data resulted in large quantities of different encryption algorithms which can be classified into two groups: symmetric key algorithms (with private key algorithms) and asymmetric key algorithms (with public key algorithms) [1]. The asymmetric key algorithm requires two different keys, one for encryption and other for decryption as shown in figure 1.

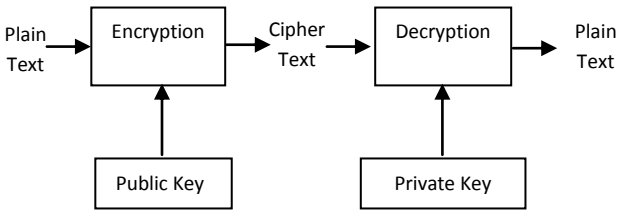

Fig. 1 Public key cryptography

The RSA algorithm is a secure, high quality, public key algorithm. It can be used as a method of exchanging secret information such as keys and producing digital signatures. However, the RSA algorithm is very computationally intensive, operating on very large (typically thousands of bits long) integers.

A vast numbers and wide varieties of works have been done on this particular field of hardware implementation of RSA encryption algorithm. A hardware implementation of RSA encryption scheme has been proposed by Deng Yuliang & Mao Zhigang. in [2], where they use Montgomery algorithm for modular multiplication. A similar approach has been taken by C. N. Zhang & Y. Xu. in [3]. This design scheme focuses on the implementation of a RSA cryptographic processor using Bit-Serial Systolic Algorithm.

This paper describes the implementation of RSA encryption/decryption algorithm on FPGA using 128 bits key size.

2. OVERVIEW OF RSA ALGORITHM

Figure 2 summarizes the different steps involved in RSA algorithm. An interesting feature of RSA algorithm is that, it allows most of the components used in encryption process are re-used in the decryption process [5]. So this can minimize the resulting hardware area.

Encryption/ Decryption

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 Key Generation

1. Choose two large primes p and q.
2. Compute $n = p \times q$
3. Calculate $\phi(n) = (p-1) \times (q-1)$
4. Select the public exponent $e \in \{1, 2, \ldots, \phi(n)-1\}$ Such that GCD $(e, \phi(n)) = 1$.
5. Compute the private key d such that $d \times e \equiv \bmod \phi(n)$

Output: public key: $k_{pub} = (n,e)$ and private key: $k_{pr} = (d)$

Fig. 2 RSA algorithm

RSA encryption and decryption are mutual inverses and commutative as shown in equation (1) and (2), due to symmetry in modular arithmetic. Hence the encryption engine covers both the operation of Encryption and Decryption.

The mathematics involved in modular arithmetic is as follows:

The integers A and B are congruent modulo m if and only if A–B is divisible by m. This congruence is written as:

$A \equiv B \bmod m$

Where m is a positive integer is called modulus.

When A and B are divided by m, the same remainder is obtained.

The sign "≡" indicates congruence.

For Examples: 14 ≡ 2 mod 12.

3. ENCRYPTION/ DECRYPTION

Encryption is the process of converting the plain text into a format which is not easily readable and is called as cipher. The conversion from plain text to cipher text involved some mathematical operation only. Hence after generation of both keys, the RSA encryption/decryption is just a modular exponentiation operation. This mathematical operation is represented as $C=M^e \bmod n$ [5], where C is cipher text, M is plain text, e is the public key exponent, and n is the modulus. This operation has involved a few modular operations: modular multiplication, modular addition, and subtraction.

3.1 Modular Exponentiation Operation

Modular exponentiation operation can further simplified in to series of modular multiplication and squaring operation. This simplification is based on an algorithm known as square and multiply algorithm. This algorithm is based on scanning the bit of the exponent from the left (the most significant bit) to the right (the least significant bit). In 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. This algorithm can be represented in pseudo code as shown in figure 3.

```
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=k-2 down to 0
C=C×C
If e_i=1 then C=C×M
```

Fig. 3 Square and Multiply Algorithm

Let us take an example exponent (e) = $(21)_{10}$
Cipher text(C) =Message (M) ^{21}
Hence binary equivalent of $(21)_{10}$ is 1 0 1 0 1 (k=5)
Where k is number of bits of exponent

For e_{k-1} =1	C=M	
For e_{k-2} =0	$C=M^2$	
For e_{k-3} =1	$C= (M^2)^2=M^4$	$C=M^4\times M=M^5$
For e_{k-4} =0	$C= (M^5)^2=M^{10}$	
For e_{k-5} =1	$C= (M^{10})^2=M^{20}$	$C=M^{20}\times M=M^{21}$

3.2 Modular Multiplication

The modular multiplication problem is defined as the computation of P = A × B (mod n), given the integers A, B, and n. It is usually assumed that A and B are positive integers with 0 ≤ A, B < n.

The square or multiplication operation is just a simple multiplication. There are many approaches to perform multiplication such as Multiply then divide, Interleaving multiplication and reduction, Brickell's method.

But in this paper Montgomery's algorithm is used. It avoids the traditional "division" operation and uses "shift and addition" operations to perform modular multiplication.

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
M = 0;
For i = 0 to k
    M=M+(A×B_i)
      if M_0 = 1
         M = M/2;
      else
      M=(M+n)/2;
    return M;
```

Fig 4: Algorithm for Modular Multiplication.

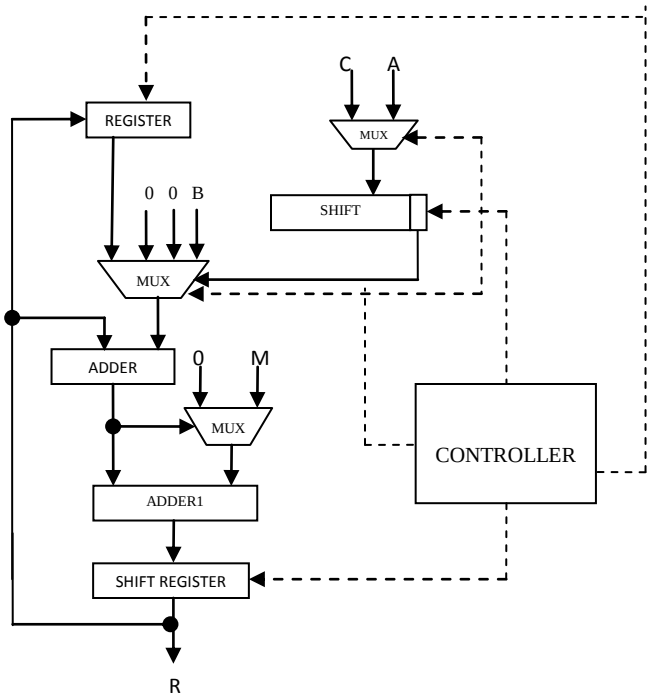

Figure 5: Montgomery's modular multiplier architecture

3.3 Addition/ subtraction

In this paper carry save adder is used to perform both addition and subtraction operation. Subtraction means addition of a number with 2's complement of other number. Here a select line is used to perform the selection of adder or subtracter. If A, B are positive integers, and S is the result then S=A+B, when select line (ADD) is 0 else S=A-B.

4. RESULT & DISCUSSIONS

The RTL schematic diagram for 128 bit encryption engine is shown in figure 6. The synthesis report for 128 bit encryption/decryption is given in Table 1-3. By changing the generic parameter; the RSA encryption module of different key size may be obtained.

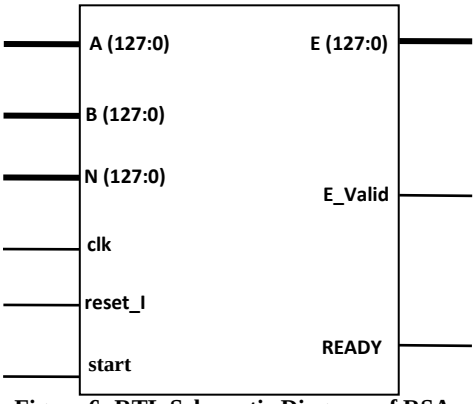

Figure 6: RTL Schematic Diagram of RSA Encryption/Decryption System.

Table 1. HDL Synthesis Report (Macro Statistics)

Component	Nos
# Adders/Subtractors	3
34-bit adder	1
34-bit subtractor	2
# Registers	430
1-bit register	403
32-bit register	7
96-bit register	4
128-bit register	16
# Comparators	1
32-bit comparator equal	1
# Xors	192
1-bit xor2	192

Table 2. Device utilization summary

	Available	Used	% of use
Selected Device	3s100evq100-4		
Number of Slices	960	2366	246
Number of Slice Flip Flops	1920	2943	153
Number of 4 input LUTS	1920	4325	225
Number of IOS	517		
Number of bonded IOBS	66	517	783
Number of GCLKS	24	1	4

Table 3. HDL Synthesis Report (Timing Summary)

Speed Grade	-4
Minimum period	9.895ns
Maximum Frequency	101.06MHZ
Minimum input arrival time before clock	6.697ns
Maximum output required time after clock	4.31
Speed Grade	-4
Minimum period	9.895ns

5. CONCLUSIONS

The VHDL code for RSA Encryption/Decryption algorithm is developed block wise. Optimized and Synthesizable VHDL code for each block synthesized using Xilinx ISE 10.1 and verified that functionally correct. The maximum clock frequency is found to ne 101.061 MHz. Since the device require more than 100% resources, it is difficult to implement in FPGA.

6. ACKNOWLEDGMENTS

We would like to express our special thank and appreciation to our parents for their support and encouragement throughout this work.

7. REFERENCES

[1] SCHNEIER, B., 1996. Applied Cryptography: Protocols, Algorithms, and Source Code in C, John Wiley & Sons.

[2] Deng Y., Mao Z., and Ye Y.,. 1998. Implementation of RSA Crypto-Processor Based on Montgomery Algorithm.

[3] Zhang. C.N, Xu. Y and Wu. C., 1997. A Bit-Serial Systolic Algorithm and VLSI Implementation for RSA.

[4] Hinek. M., 2010. Cryptanalysis of RSA and Its Variants.

[5] Rivest, R., Shamir, A., and Adleman, L, 1978. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. Communications of the ACM.

[6] Stallings W.2003, Cryptography and Network Security: Principles and Practices.

[7] Burnett S. and Paine S, 2001. RSA Security's Official Guide to Cryptography. McGraw-Hill.

[8] Ashenden P. and Lewis J, 2006. The Designer's Guide to VHDL. Morgan Kaufmann Publishers.

[9] Hwang E. Digital Logic and Microprocessor Design with VHDL.

[10] Nedjah.N and Mourelle L.2002.Two Hardware Implementation for the Montgomery Modular Multiplication: Sequential versus Parallel. IEEE.