Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 165 (2019) 525–531

Procedia Computer Science

www.elsevier.com/locate/procedia

INTERNATIONAL CONFERENCE ON RECENT TRENDS IN ADVANCED COMPUTING 2019, ICRTAC 2019

# Fast and Area Efficient Implementation of RSA Algorithm

Sheba Diamond Thabah, Mridupawan Sonowal, Rekib Uddin Ahmed, Prabir Saha*

*National Institute of Technology Meghalaya, Shillong-793003, India*

**Abstract**

Efficient hardware implementations of public-key cryptosystems have been gaining interest in the past few decades. To achieve the goal, a high frequency as well as low latency Rivest-Shamir-Adleman (RSA) cryptosystem is reported in this paper. To configure such cryptosystem shift-add multiplier have been re-constructed and binary digit based modular exponentiation circuitry is proposed. Such exponentiation circuitry has been implemented through binary bit distribution technique, where, most significant bit (MSB) has been discarded for the implementation, owing to increase the operating frequency. The functionality of the reported algorithms were justified and compared in Hardware Description Language (HDL), simulated in Modelsim and synthesized in Xilinx ISE 14.2 platform. The proposed hardware implementation of RSA algorithm has a maximum frequency of operation of 545 MHz and 298 MHz for the bit sizes of 8 and 64 respectively. The proposed method shows improvements in terms of speed as well as in number of Look-up-tables (LUTs). Moreover, application-specific integrated circuit (ASIC) implementation of such cryptosystem of RSA was carried out through Encounter® RTL Compiler v11.10-p005_1 of Cadence® tool.



*Keywords:* Application specific integrated circuit (ASIC); binary methodology; modular exponentiation; Rivest-Shamir-Adleman (RSA) algorithm.

## 1. Introduction

Nowaday's internet or electronic communication has become an important and unavoidable part of our daily life [1]. Moreover, online banking, shopping, electronic-mail (e-mail), voting etc., have the requirement of security.

* Corresponding author. Tel.: +91-0364-2501294; fax:+91- 0364-2501113
*E-mail address:* sahaprabir1@gmail.com


10.1016/j.procs.2020.01.024

Therefore secure communication system plays a pivotal role in the present era [1-3], where the security schemes would be provided by cryptosystem.

In general, cryptosystems can be classified into two groups [1-4], viz., symmetric key cryptosystem also called private-key cryptosystem and asymmetric key cryptosystem which are usually called the public-key cryptosystem [1-9]. Since the advent of asymmetric key cryptography, originally proposed by Diffie and Hellman [2] in 1976, the security of systems and communication became much stronger [4]. Cryptosystems achieve stronger security by the application of modular arithmetic algorithms, like multiplication and exponentiation [4-9], and by suppressing the symmetrical keys. Asymmetric cryptosystem has two different keys for the encryption and decryption algorithm [10, 11]. These two keys have to be generated or shared between the sender and the receiver. The algorithm can be implemented either in software or hardware. Both types of implementation has some certain pros and cons, software version has some advantages like of ease of use, possibility of up-gradation, flexibility, portability etc. [12], whereas, hardware implementation has more physical security as it is not dependent on the operating system and also could decrease time consumption leading to improved time efficiency [12].

The work in this paper focuses on the hardware implementation of a widely used asymmetric cryptographic algorithm; Rivest-Shamir-Adleman (RSA) algorithm [5]. To implement the RSA algorithm in hardware, it requires different arithmetic blocks like modular multiplication and exponentiation [5-11], etc. Various designs have been proposed to achieve the hardware implementation goals (either speed improvement or area reduction). To achieve speed improvement, repetitive modular multiplication and modular exponentiation have been proposed in [13]. Modular exponentiation requires repetitive computation of the modular product, therefore modular exponentiation over a large modulus becomes difficult, resulting in speed reduction [5].

In [5, 7], Montgomery algorithm has been adopted for modular multiplication and exponentiation, which has an issue of carrying propagation. In [6], the redundant representation is considered for implementation, with the penalty of conversion algorithm. In [9], two new architectures have been designed for Montgomery algorithm where the latency is improved by half of the original architecture using parallelism through the pre-computation of the partial results, with improvement in area consumption. However, long multiplication chain is required in the Montgomery multiplication algorithm, leading to efficiency reduction in RSA. To achieve the area improvement, researches rely on the shift-add algorithm for multiplication [14].

In this paper, application specific integrated circuit (ASIC) implementation of the RSA algorithm is reported. The multiplier has been restructured and new circuit modules for modular exponentiation have been proposed to achieve better results. Shift-and-add technique for multiplication has been reconstructed and a new hardware implementation methodology has been proposed for the computation of modular exponentiation, through binary bit distribution. Algorithmic implementation has been carried out, simulated and synthesized in different Xilinx ISE 14.2 platform for the RSA cryptosystem. Moreover, ASIC implementation has been carried out by Encounter® RTL Compiler v11.10-p005_1 of cadence tool. Performance parameters in algorithmic level as a function of the look-up tables (LUTs), the maximum frequency of the operations have been reported and compared with the existing architectures. Similarly, performance parameter metrics for ASIC implementation have been measured and compared as a function of propagation delay, power consumptions, gates count and area measurement. The implemented RSA algorithm has the maximum frequency of operation as 298 MHz for 64-bits, which is faster from the earlier reported ones.

## 2. RSA algorithm

RSA is a public key cryptographic algorithm. The major steps involve in RSA algorithm [14, 15] are

- Key generation
- Encryption process
- Decryption process.

Module wise description of the above mentioned algorithm is described in the subsequent paragraphs.

### *2.1. Key generation*

To generate the public key and private key [14, 15], algorithm 1 is followed. In this algorithm (algorithm 1),$p_1$, $p_2$ are the selected inputs, and $e$, $d$ are the outputs which are the two keys required for the RSA algorithm. To generate

these two keys, select two large prime numbers of equal length, i.e., $p_1$ and $p_2$. Compute the modulus number $n$ and the function $\Psi(n)$. Choose a positive integer $e$ such that,$1 < e < \Psi(\mathrm{n})$, calculate $\gcd(e, \Psi(\mathrm{n})) = 1$. Here, $e$ is called the encryption key or the public key. Then, compute the decryption key or the private key$d, 1 < d < \Psi(\mathrm{n})$, such that,$e.d \bmod \Psi(n) = 1$. The decryption key $d$ is obtained by taking the multiplicative inverse of the encryption key $e$.

**Algorithm 1:** Key Generation

| | |
|---|---|
| Input: | $p_1, p_2$ |
| Output: | $e, d$ |
| 1: $n = p_1 * p_2$; | |
| 2: $\Psi(n) = (p_1 - 1)(p_2 - 1)$; | |
| 3: $1 < e < \Psi(\mathrm{n}), \gcd(e, \Psi(\mathrm{n})) = 1$; | |
| 4: $e.d \bmod \Psi(n) = 1$; | |

THREE MAIN functions to write:
1. Key generation
2. encryption process
3. decrytion process

## 2.2. Encryption process

The RSA encryption process [15] can be computed through algorithm 2, where modular exponentiation has to be performed by the sender. First, get the public key $(e, n)$ of the recipient and also the plaintext to be sent which is represented as $m$. After receiving $e$, $n$, $m$, encrypt the plain text into cipher-text or coded text and then transmit the cipher-text through the channel to the recipient.

**Algorithm 2:** Encryption process

| | |
|---|---|
| Input: | $m, n, e$ |
| Output: | $m^e \bmod n$. |
| 1: $c = m^e \bmod n$; | |

## 2.3. Decryption process

The decryption process [15] in algorithm 3 has to perform the reverse operation of encryption process. The decryption process has to be performed by the receiver using his private key $(d, n)$ to get the original plaintext. To do so, decrypt the cipher-text received from the sender by computing the modular exponentiation of $c$ with respect to the modulus $n$.

**Algorithm 3:** Decryption process

| | |
|---|---|
| Input: | $c, n, d$ |
| Output: | $c^d \bmod n$. |
| 1: $m = c^d \bmod n$; | |

# 3. Hardware implementation of RSA algorithm

Don't need this part… not doing hardware implementation

The hardware implementation of RSA algorithm requires modular multiplication and exponentiation circuits. In this paper, circuit level modification has been carried out to achieve high speed. Multiplier circuit was revisited and modulo exponential circuitry is being proposed.

## 3.1. Multiplier

The multiplier shown in Figure 1 is based on add and shift operation. Add and shift multiplier is better than other multiplier like Montgomery multiplier because of its lesser complexity and smaller resources consumption [14]. This multiplier is reconstructed through algorithm 4, where shifting operation is performed for the multiplicand and the multiplier rather than the partial product stored in $temp$ register. In this way, the multiplier operation is reduced by one clock cycle, as in the normal shift add multiplier the shifting is done after addition, while in this restructured multiplier the multiplicand and the multiplier are shifted while addition is perform.

Here, the inputs or the operand to be multiplied are first loaded into registers $temp1$ and $temp2$ as shown in Figure 1. The circuit consists of three registers, an adder, a control logic and shift logic. The operation is shown for 4-bit multiplication. First, the multiplicand and multiplier are loaded into a 2n- bit registers, the four least significant bits are the multiplicand and the multiplier values and the rest are appended with 0's. Initially $temp$ registers are loaded with 0's. The control logic block is used to check the values of multiplier, and shift logic block used for shifting the multiplier and multiplicand right and left respectively.

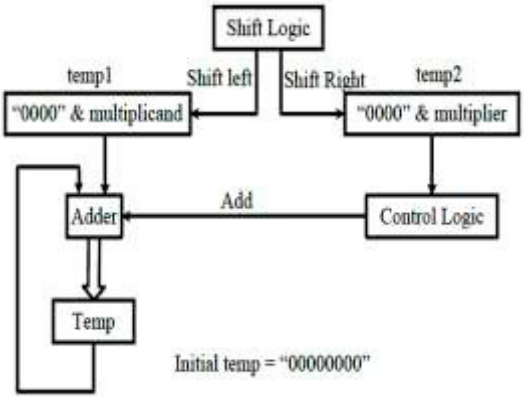


Fig. 1. Multiplier circuit of 4-bits.

**Algorithm 4:** Modified multiplier

| | |
|---|---|
| Input: | $multiplicand, multiplier$. |
| Output: | $temp$. |

1. $temp := 0$;
2. $temp1 := multiplicand$;
3. $temp2 := multiplier$;
4. For $i = n - 1$ downto 0 then
   If $temp2(0) = 1$ then
   $temp := temp1 + temp$;
   $temp1 := temp1 \ll 1$;
   $temp2 := temp2 \gg 1$;
   Else
   $temp1 := temp1 \ll 1$;
   $temp2 := temp2 \gg 1$;
   End if;
   End for;
5. Return $temp$

### *3.2. Modular exponentiation*

Exponentiation operation has a vital role in RSA algorithm [9, 16]. The encryption and the decryption process in RSA are based on modular exponentiation [9, 13]. There are different types of modular exponentiation algorithm, but the most used algorithm in RSA implementation is the square and multiply algorithm [17], as this algorithm reduces the problem of long carry propagation to a series of modular multiplication and squaring steps. Here the square and multiply algorithm of implementing modular exponentiation is restructured and proposed. In this proposed algorithm of modular exponentiation, the speed increases as compared to the existing square and multiply algorithm because one bit is deleted for the operation, which reduces the number of clock cycles for operation.

The algorithm 5 describes the proposed modular exponentiation algorithm. If $a$ and $b$ are two positive integers, then for a given positive modulus $n$, the modular exponentiation is $c = a^b \ mod \ n$. Firstly, convert the exponent value into binary value and delete the most significant bit from the obtained binary value. Take the remaining binary value for calculation, and the operation is done from the left side and moves to the right side. When binary '1' appears,

square operation of the given base number is performed followed by multiplication. When, binary '0' appears, only squaring operation of the previous result is performed.

**Algorithm 5:** Proposed Modular Exponentiation

| | |
|---|---|
| Input: | $a, n, b = (b_{n-1}, \dots, b_1, b_0)_2$. |
| Output: | $a^b \ mod \ n$. |

1. $b = (b_{n-1}, \dots, b_1, b_0)_2$;
2. $b = (b_{n-2}, \dots, b_1, b_0)_2$;
3. $c = a$;
4. For $i = n - 2$ downto 0 do
   If $bi = 1$ then
   $c := c \ . \ c \ mod \ n$;
   $c := c \ . \ a \ mod \ n$;
   Else
   $c := c \ . \ c \ mod \ n$;
   End if;
   End for;
5. Return $c$

## 4. Results and discussions

The implementation methodology has been restructured to achieve the high-frequency operation. The main time-consuming part of the RSA cryptosystem such as modulo multiplication and modulo exponentiation has been proposed in this paper. The proposed algorithm has been incorporated in the RSA cryptosystem and synthesized the algorithm. The code of the RSA algorithm is written in VHDL, simulated in Modelsim and synthesized in Xilinx ISE 14.2 platform. For synthesizing purpose, Virtex 6 (xc6vlx760-2ff1760), Virtex 6 Low Power (xc6vlx760l-1lff1760) and Virtex 7 (xc7v2000t-2flg1925) devices have been chosen. For ASIC implementation, Encounter® Compiler v11.10-p005_1 of Cadence® has been chosen.

Table 1. Device utilization report of RSA algorithm.

| Performance Parameters | Device | 8-bits | 16-bits | 32-bits | 64-bits |
|---|---|---|---|---|---|
| Number of Slice LUTs | Virtex Series | 2137 | 7631 | 28K | 109K |
| Maximum Frequency (MHz) | Virtex 6 | 532 | 465 | 371 | 264 |
| | Virtex 6 Low Power | 537 | 476 | 387 | 282 |
| | Virtex 7 | 545 | 488 | 403 | 298 |

Table 2. Post synthesis result of the RSA algorithm obtained using encounter® compiler v11.10-p005_1 of cadence tool (library:osu018_stdcells).

| Performance Parameters | 8-bits | 16-bits | 32-bits | 64-bits |
|---|---|---|---|---|
| Delay (ps) | 30672.8 | 119307.6 | 440823 | 865882 |
| Power (nW) | 7857.3 | 12067.7 | 24692.1 | 59326.2 |
| Gates | 6977 | 21139 | 75246 | 469813 |
| Area ($\mu m^2$) | 203920 | 683165 | 2527486 | 12565280 |

The synthesized results of RSA algorithm is shown in Table 1. The device utilization in terms of LUTs and frequency (in MHz) is tabulated for bit lengths of 8, 16, 32, and 64 respectively. The synthesized frequency of RSA algorithm is around 532, 465, 371, and 264 MHz for bit sizes of 8, 16, 32, and 64 respectively when implementing in Virtex 6 (xc6vlx760-2ff1760) device. The synthesized frequency obtained for the Vertex 6 Low Power (xc6vlx760l-1lff1760) device is around 537, 476, 387 and 282 MHz for bit sizes of 8, 16, 32, and 64 respectively due to the variation of the architecture of the device. Similarly, 545, 488, 403 and 298 MHz are the maximum operating frequency when using Virtex 7 (xc7v2000t-2flg1925) device for the bit sizes of 8, 16, 32, and 64 respectively. The

LUTs consumption of the proposed architecture is 2137, 7631, 28285 and 108937 for bit sizes of 8, 16, 32, and 64 respectively.

Table 2 shows the ASIC implementation results of the proposed architecture for the RSA encryption and decryption process for comparing delay (in ps), power (in nW), number of gates and area (in $\mu m^2$) using Encounter® RTL Compiler v11.10-p005_1 of Cadence® tool. The performance parameters of the RSA algorithm are tabulated for bit sizes of 8, 16, 32, and 64 respectively. The delay (in ps) taken by RSA algorithm for bit sizes of 8, 16, 32, and 64 are 30672.8, 119307.6, 440823 and 865882 respectively and the power (in mw) consumed are 7.85, 12.06, 24.69 and 59.32 respectively. The number of gates used are 6977, 21139, 75246 and 469813 and the total area (in μm2) taken by the proposed architecture are 203920, 683165, 2527486 and 12565280 for bit sizes of 8, 16, 32, and 64 respectively.

Table 3 gives the comparison of the existing architecture with the proposed architecture in implementing the RSA algorithm. The architecture in [14], used add-and-shift algorithm for modular multiplication, and used square-and-multiply algorithm for modular exponentiation. It has been observed that the operating frequency was 100MHz in Xilinx Virtex II Pro FPGA device. In architecture [18], the RSA encryption and decryption process used Montgomery multiplier for modular multiplication and square-and-multiply algorithm for modular exponentiation, the frequency of operation was 54.7MHz in Altera Apex20KE (EP20K200EBC356) device. In architecture [19], square-and-multiply algorithm with right-to-left mechanism is used where the synthesized frequency obtained is 58MHz using Virtex 5. The proposed architecture has a frequency of 545 MHz and 298 MHz for the bit sizes of 8 and 64 respectively. From Table 3, it has been observed that the frequency of operation in the proposed implementation has been increased from the earlier reported architectures listed in Table 3 for implementing RSA cryptosystem architecture. Thus, the proposed implementation is faster than the earlier implementations listed in Table 3.

Table 3. Comparison of the results with the previous works.

| Architecture | Bit Size | Frequency (MHz) |
|---|---|---|
| [14] | 8 | 100 |
| Proposed | 8 | 545 |
| [18] | 64 | 54.7 |
| [19] | 64 | 58 |
| Proposed | 64 | 298 |

## 5. Conclusions

RSA algorithm with a higher frequency of operation has been implemented and reported in this paper. To achieve the higher frequency of operation, the central part of the algorithmic module i.e., modulo exponentiation circuitry has been proposed. The RSA algorithm has been implemented for different encryption key lengths of 8, 16, 32 and 64 bits. The behavioral implementation of RSA is done using Xilinx 14.2 tool. The maximum frequency obtained for running the RSA cryptosystem for 8-bits and 64-bits is 545 MHz and 298 MHz respectively which is faster from earlier reported ones. Moreover, ASIC implementation has been carried out through Encounter® Compiler of Cadence.

## References

[1]. Hong, Jin-Hua, and Cheng-Wen Wu. (2003) “Cellular-Array Modular Multiplier for Fast RSA Public-Key Cryptosystem Based on Modified Booth’s Algorithm.” *IEEE Transactions on VLSI Systems***11** (**3**): 474-484.

[2]. Diffie, Whitfield, and Martin E. Hellman. (1976) “New directions in cryptography.” *IEEE Transactions on Information Theory***22** (**6**): 644-654.

[3]. Goldreich, Oded. (2004) “Foundations of Cryptography: Basic Applications”, in Vol. 2, Cambridge University Press.

[4]. Stinson, Doug. R. (2002) “Cryptography: Theory and Practice”, in 2nd ed. Chapman & Hall/CRC, Boca Raton.

[5]. Shieh, Ming Der, Jun-Hong Chen, Hao-Hsuan Wu, and Wen-Ching Lin. (2008) “A New Modular Exponentiation Architecture for Efficient Design of RSA Cryptosystem.” *IEEE Transactions on VLSI Systems***16** (**9**): 1151-1161.

[6]. Takagi, Naofumi, and Shuzo Yajima. (1992) “Modular Multiplication Hardware algorithms with a Redundant Representation and their Application to RSA Cryptosystem.” *IEEE Transactions on Computers***41** (**7**): 887-891.

[7]. Yang, Ching-Chao, Tian-Sheuan Chang, and Chien-Wei Jen. (1998) "A New RSA Cryptosystem Hardware Design Based on Montgomery's Algorithm." *IEEE Transactions on Circuits and Systems-II***45** (**7**): 908-913.

[8]. Homma, Naofumi, Atsushi Miyamoto, Takafumi Aoki, and Adi Samir. (2010) "Comparative Power Analysis of Modular Exponentiation Algorithms." *IEEE Transactions on Computers***59** (**6**): 795-807.

[9]. Huang, Miaoqing, Kris Gaj, Tarek El-Ghazawi. (2011) "New Hardware Architectures for Montgomery Modular Multiplication Algorithm." *IEEE Transactions on Computers***60** (**7**): 923-935.

[10]. Rivest, Ron, Adi Shamir, Leonard Adleman. (1978) "A method for obtaining digital signatures and public-key cryptosystems." *Communication of the ACM***21** (**2**): 120-126.

[11]. Shand, Mark, and Jean Vuillemin. (1993) "Fast Implementation of RSA Cryptography." In *Proceedings of IEEE 11th Symposium on Computer Arithmetic*, Windsor, Ontario, Canada, June 1993, 252-257.

[12]. AlKalbany, Abdullah, Hussein Ahmad AlHassan, and Magdy Saeb. (2005) "FPGA Implementation of the Pyramids Block Cipher." In *Proceedings 2005 IEEE International SOC Conference (SOCC2005)*, Herndon, VA, 2005, 271-275.

[13]. Morita, Hikaru. (1989) "A fast modular-multiplication algorithm based on a higher radix." in Brassard G. (eds) *Advances in Cryptology-CRYPTO' 89 Proceedings*, CRYPTO 1989, Lecture Notes in Computer Science, vol 435, Springer, New York, NY, 387-399.

[14]. Rahman, Mostafizur, Iqbalur Rahman Rokon, and Miftahur Rahman. (2009) "Efficient hardware implementation of RSA cryptography." In *2009 3rd International Conference on Anti-counterfeiting, Security and Identification in Communication*, Hong Kong, China, 06 October 2009, 316-319.

[15]. Aswathy, B. G., and R. Resmi. (2014) "Modified RSA Public Key Algorithm." In *2014 1st International Conference on Computational Systems and Communication (ICCSC)*, Trivandrum, 17-18 Dec. 2014, 252-255.

[16]. Okamoto, Tatsuaki, and Shigenori Uchiyama. (1998) "A New Public-Key Cryptosystem as Secure as Factoring." in Nyberg K. (eds) *Advances in Cryptology -EUROCRYPT'98*, EUROCRYPT 1998, Lecture Notes in Computer Science, vol 1403, Springer, Berlin, Heidelberg, 308-318.

[17]. Knuth, Donald. E. (1981) "The Art of Computer Programming", in Vol. 2 of Seminumerical Algorithm, 2nd ed. Reading, MA: Addison-Wesley.

[18]. Rajavelu, Srinivasan, V. Vaidehi, J. Balaji, and S. Heema. (2011) "Single Chip Efficient FPGA implementation of RSA and DES for Digital Envelop Scheme", Madras Institute of Technology Campus, Anna University, Chennai-600044, India, 2011.

[19]. Shams, Rehan, Fozia Hanif Khan, and Mohammad Umair. (2013) "Cryptosystem an Implementation of RSA Using Verilog." *International Journal of Computer Networks and Communications Security***1** (**3**): 102-109.