International Journal of Engineering & Technology, 7 (3.24) (2018) 278-283

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

ASIC Design of Low Area RSA Crypto-core based on Montgomery Multiplier

Richard Boateng Nti, Kwangki Ryoo*

Department of Information & Communications Engineering, Hanbat National University, Daejeon 34158, Republic of Korea
Corresponding author E-mail: kkryoo@gmail.com

Abstract

Background/Objectives: Currently, the most popular public key encryption is RSA. We present a low area hardware design of RSA crypto-core based on the Montgomery algorithm in ASIC.
Methods: We employed Carry Save Adder in the design of the multiplier which plays a critical role in the overall design. The proposed hardware was designed at Register Transfer Level using Verilog description language with Xilinx 14.3 ISE design suite and simulated with ModelSim. In ASIC implementation, TSMC 90nm and 130nm CMOS technology was employed for synthesis of the Montgomery multiplier and modular exponentiation respectively.
Findings: The primal operation of RSA cryptosystem is modular exponentiation, computed by repetitions of modular multiplications. Fast modular multiplication algorithms over the years have been proposed to speed up exponentiation and yet maximize performance. The core of this paper evolved from the modification of the modified Montgomery multiplication algorithm. From the new algorithm, a hardware architecture which simplifies the operation of the Q_logic coupled with a compact two-level CSA in the Montgomery multiplier is designed. The simplified Q_logic design and the elimination of traditional BRFA and bypass circuitry accounted for a reduction in area. Furthermore, the new multiplier is applied in the H-algorithm to develop the modular exponentiation unit. Other relevant modules in the RSA crypto-core including the pseudorandom number generator, primality tester and key generator have been optimized for resource sharing to balance and improve speed and area of the system. Synthesis results of the proposed multiplier and exponentiation unit achieved a gate count of 60K and 79K representing a reduction of 47% and 28% respectively.
Improvement/Applications: Our system is suitable for low area RSA applications. Future works on this paper will examine the analysis and design of Carry Save Adder to improve propagation delay.

Keywords: *Public key cryptosystem, Carry Save Adder (CSA), RSA, Montgomery multiplication, Barrel Register Full Adder (BRFA), Modular exponentiation.*

1. Introduction

Smart devices such as smartphones, tablets and phablets, smartwatches etc., PCs and the Internet infrastructure provide numerous benefits, which make living comfortable. A major factor of this technological advancement is information processing: gathering, keeping, transmitting and analyzing data. Upkeep of data privacy and integrity is essential from the perspective of security in this age of digital communication. Data integrity, authentication, nonrepudiation and confidentiality are vital in public key cryptosystem. The private and public keys in public key cryptography are generated by either the user or a key generator [1]. Modular exponentiation with large numbers in PKC such as Diffie-Hellman and RSA [2] is the most time-consuming operation and can be computed repeatedly using modular multiplication [3]. To speed up decryption and encryption in RSA, efficient modular multiplier design is needed.

There are two techniques of designing modular multiplier: the systolic approach and the Montgomery approach. The Montgomery multiplication algorithm [4] presented in 1985 proves to be the most efficient technique, especially for hardware implementation. The Montgomery algorithm is implemented in two methods. The intermediate results in the first method [5,6] are kept in carry save format to do away with propagation(carry) while the input and output operands remain in binary representation. Final conversion to binary representation from the carry save format must be performed. This comes with an extra cost of using the CPA [5]. Consequently, there is an increment in the overall computation time which implies logically that the operands length directly affect the throughput.

Contrary to the aforementioned approach, some works utilized 5-to-2 CSA [7,8], a three-level CSA tree, to solve this issue without the need for format conversion. This second approach, excluding the final step for the results, eliminates repeated interim output of the Montgomery modular multiplication output to input conversion by maintaining all input and output operands in the carry save format. This leads to more registers and CSA for the increase in operands but fewer clock cycles. Mclvor et al. [8] suggested two algorithmic versions of the radix-II Montgomery multiplication algorithm with both approaches using CSA. One is based on a 5-to-2 CSA while the other on a 4-to-2 CSA with a multiplexer. Given that k is the bit length of the operand, both variants proposed can perform multiplication in k+1 and k+2 clock cycles respectively.

The intention of this work is to design a complete RSA cryptosystem from key generation to encryption/decryption with the exponentiation unit based on the proposed Montgomery multiplier. The multiplier design employs a two-level adder and a

simplified Q_logic for bit operation which accounts for a reduction in the hardware area. The new Montgomery design embedded into the R-L modular exponentiation algorithm with efficient hardware structure played a role in enhancing performance. In addition, resource sharing of modules among the key generator, primality and exponentiation unit ensured area maximization. Section 2 of this paper reviews the general concept of RSA and the various algorithms involved. We propose a low area hardware design of the RSA crypto-core in section 3 with modular multiplication and exponentiation as the foundation. In Section 4, we check the dissimilarity or similarity between different Montgomery multiplier designs as well as modular exponentiation units. Lastly, Section 5 concludes the paper.

2. RSA cryptosystem

The Rivest-Shamir-Adleman algorithm [2] generally called RSA crypto scheme is the most popular asymmetric key encryption. Figure 1 gives a pictorial view of the PKC process. There exit several applications of RSA but the notable ones are key sharing i.e. encrypting small piece of information and digital signatures. The integer factorization problem is the one-way function for RSA cryptosystem. It is mathematically easy to evaluate the multiplication of two large numbers. However, the other way around is extremely difficult. Mathematical functions such as Euler's phi and Euler's theorem are vital in carrying out the computation of RSA. Basically, the operation of RSA cryptosystem is categorized into key derivation, encryption and decryption.

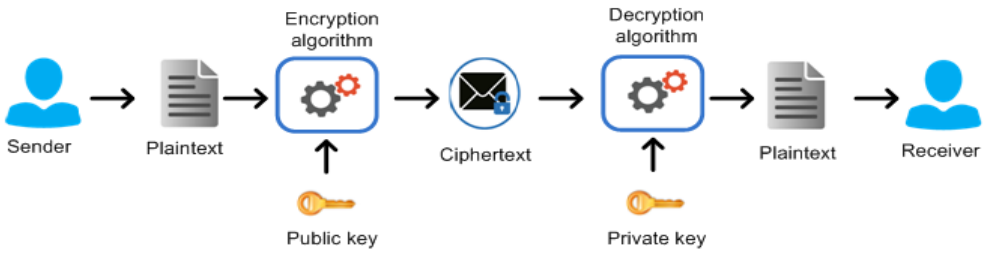

Figure 1: Architecture of public-key cryptography

Key generation: It is the setup phase where two distinct keys (public and private) are derived. Algorithm 1 describes the step by step process for key generation.

Algorithm 1: Key Generation (RSA)
Input: s and r
Output: n, e and d
1. Generate two large distinct prime numbers s and r
2. Calculate $n = s * r,\ \emptyset = (s-1) * (r-1)$
3. Choose a public exponent $e \in \{1,2,\ldots \emptyset(n)-1\}$ such that $\gcd(e, \emptyset(n)) = 1$
4. Using extended Euclidean algorithm, calculate $1 < d < \emptyset$ such that $e * d \equiv 1 \bmod \emptyset$

Encryption: Given public key $kpu = (n, e)$ and plaintext a, encryption is computed as:

$$b = e_{kpub}(a) \equiv a^e \bmod n \quad \text{where } a, b \in Z_n$$

Decryption: Given private key $kpr = (d, n)$ and ciphertext b, decryption is evaluated as:

$$a = d_{kpr}(b) \equiv b^d \bmod n \quad \text{where } a, b \in Z_n$$

In the key generation, the letters n, d and e are known as the modulus, decryption exponent and encryption exponent respectively. Both the decryption and the encryption operations are done in the integer ring and they determine the complexity of the RSA cryptosystem. As a result, more emphasis will be laid on modular exponentiation and multiplication in this section.

2.1. Modular Exponentiation Overview

Modular exponentiation is usually evaluated by executing modular multiplications repeatedly [3]. Thus, many algorithms such as the m-ary method, the Booth recording method and the adaptive m-ary method have been proposed [9]. The concept behind this research into modular algorithm is to decrease the number of times multiplication is executed. The Binary method among the lot is the most efficient approach for computing modular exponentiation. Therefore, it is considered as the de facto for exponentiation. The rudimentary idea of the binary method lies in observing the exponent bit-wise and computing squaring and multiplication until all bits of the exponent is exhausted. Two methods of this binary scheme exist: L-R method and R-L method. These two are distinct from each other from the starting points of operation. The L-R technique, from the LSB to the MSB, offers parallelism capabilities in the multiplication and squaring sacrificing area for speed while the other is vice-verse [10]. The R-L technique is known as the H-algorithm

The exponentiation algorithm is applied to calculate $M^E \bmod N$. Let M be a k-bit message and E indicates a k_e–bit exponent. Algorithm 2 depicts the R-L technique where scanning of the exponent is done from the most significant bit. The two constant factors K and R are usually calculated in advance and the value of R is either 2k or 2k+1. ModMul refers to modular multiplier.

Algorithm 2: H-Algorithm
Input: E, M, N, $K \equiv R^2 \bmod N$
Output: $Z = M^E \bmod N$
```
1  P = ModMul(M, K, N), Q = ModMul(K, 1 ,N);
2  for i = k_e - 1 to 0 {
3     Q = ModMul(Q, Q, N);
4     if(E_[i] = 1)
5        Q = ModMul(P, Q, N);  }
6  return Z =ModMul(Q, 1, N);
```

2.1.1. Modular Multiplication Overview

The Montgomery multiplication algorithm [4] is used to evaluate the multiplication of two integers X and Y modulo N. Algorithm 3 presents the original version of the algorithm in radix-II. Given two integers x and y; where x, y < N (n-bit modulus), the value of $R = 2^n \bmod N$ and bounded by $2^{n-1} \leq N < 2^n$. The N-residue of integers x and y with regards to R is defined as equation (1). From equation (1), the modular product S of X and Y is derived as equation (2) given that R^{-1} is the modulo inverse of R and N that is $R^{-1} * R = 1 \bmod N$. Since the convergence variation between upper and lower bounds of S in algorithm 3 is given as $0 \leq S < 2N$, an extra operation $S = S - N$ is needed to get rid of the oversized residue whenever $S \geq N$. The computation of S on line 4 determines the critical delay in algorithm 1.

$$X = (x * R)\ mod\ N,\quad Y = (y * R)\ mod\ N \tag{1}$$

$$S = (X * Y * R^{-1})\ mod\ N \tag{2}$$

Algorithm 3: Radix-II Montgomery Multiplication
Input: X, Y, N
Output: S[n]
1. Initialize S[0] = 0;
2. for i = 0 to n - 1 {
3. qi = (S[i] + Xi ∗ Y_0) mod 2;
4. S[i+1] = (S[i] + Xi ∗ Y + qi ∗ N) / 2;}
5. if(S[n] ≥ N)
6. S[n] = S[n] - N;
7. return S[n];

Several modifications have been made to improve the Montgomery algorithm. Walter [11] presented a fast Montgomery method with no final subtraction and comparator. The idea presented in paper [11] shown as algorithm 4, describes a process which sacrifices two extra cycles to achieve the desired results to obtain minimal area. In other proposals, efforts have been made [12,13] to reduce the critical path delay and eliminate superfluous operations. Kuang el al. [13] proposed a high-throughput architecture using Carry Save Adder and extra circuitry that bypass superfluous operation. The number of cycles drastically reduced especially in situations of many bands of consecutive zeros.

Algorithm 4: Modified Montgomery Multiplication [11]
Input: X, Y, N
Output: S[n+2]
1. Initialize S[0] = 0;
2. for i = 0 to n + 1 {
3. qi = (S[i] + Xi ∗ Y_0) mod2;
4. S[i+1] = (S[i] + Xi ∗ Y + qi ∗ N) / 2; }
5. return S[n + 2]

2.2. Overview of Pseudorandom Number Generation

When sequence of symbols or numbers cannot be easily anticipated or predicted by chance, the phenomenon is called random number generation. Different methods of random number generations have been presented based on the application area. Generally, there are two types of random number creation namely true and pseudo-random number generation. The formal approach usually relies on physical phenomenon or natural occurrence e.g. thermal noise, entropy from cosmic radiations, atmospheric noise etc. Contrarily, a pseudo-random number generator generates a sequence a numbers form an algorithm with an initial value called seed. The Akari_1 [14] described as algorithm 5 is an PRNG of concern.

Algorithm 5: Akari_1
Input: Seed
Output: b
1. Initialize a0 and a1 (where a0 and a1 are of size n-bits)
$a0 = a0 + ((a0 * a0) \nabla 5);$
$a1 = a1 + ((a1 * a1) \nabla 13);$
$b = a0;$
2. for i from 0 to 63 {
$b = (b \gg 1) + (b \ll 1) + b + a1;$ }
3. return lower half of b (n/2 bits);

2.3. Primality Testing

A primality test is an algorithm run on a given number to ascertain whether it is prime or not. Assuming p is a prime number, $p - 1 = 2^k m$ where m is odd. Let $1 \leq a < p - 1$. One of the equations below must hold;

I. $a^m = 1 \; mod \; p$

II. One of $a^m, a^{2m}, a^{4m}, \ldots \; a^{2^{k-1}m} \equiv -1 \; mod \; p$

Algorithm 6: Miller-Rabin [15,16]
1. Randomly select $a_1, \ldots, a_k$ independent number $1 < a < p - 1$.
2. For each element of $a(a_1, \ldots, a_k)$ test for I and II
3. If any a_1 is a witness that p is composite, p is not a prime number.
4. If there is no a_1 as a witness, then p is probably a prime.

3. Proposed RSA hardware design

The RSA design proposed presents a complete RSA cryptosystem which includes a prime number generator, key generation and the encryption/decryption process. The basic process for the key generation and modular exponentiation was described in section 2. The functional operation of this module is to generate a key-pair and encrypt or decrypt a message. The initial process is to generate keys which can only be possible after the prime number generation is done. Following the key generation is the actual RSA operation. Based on the control signal given either decryption or encryption is executed with the private or public key respectively. In figure 2, the general structure of the proposed design is illustrated.

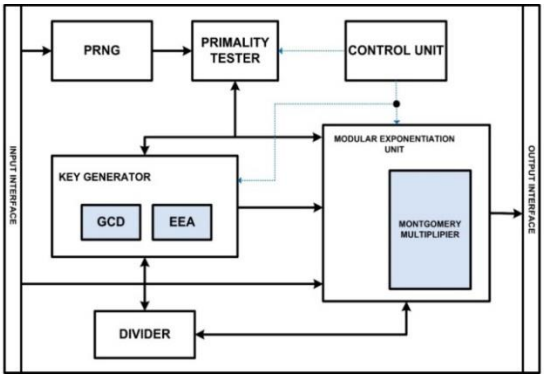

Figure 2: Proposed RSA cryptosystem hardware structure

3.1. Modular Exponentiation Design

Our propose hardware design employs the R-L approach because area-optimization is the goal of this research work. The algorithm is as shown in algorithm 2. Given the inputs M, E, N and K representing the message, exponent i.e. public or private, modulus and a constant factor respectively, the proposed structure as shown in figure 3 computes the algorithm. The Montgomery multiplier is repeatedly applied throughout the entire exponentiation.

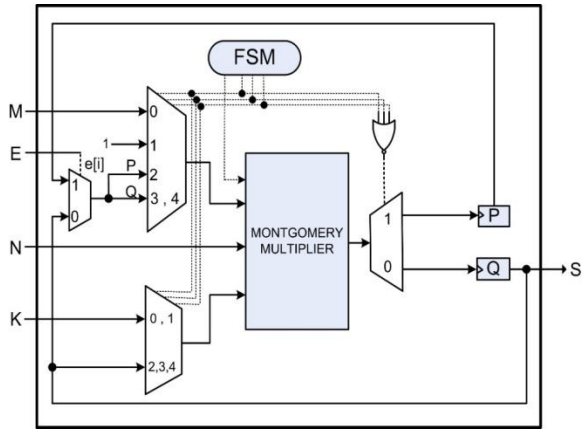

Figure 3: Architecture of the proposed exponentiation design

The initial step in algorithm 2 is the pre-processing phase which computes Q and P. The finite state machine (FSM) shown in figure 3 dictates the path of data in and out of the Montgomery multiplier. For state 0, M and K are directed to the ModMul; the output from the multiplier goes to the register P. The gate combination after the multiplier ensures that P gets the value for the first evaluation. State 1 follows suit with 1 and K being routed

to the ModMul design and then stored in Q. The system iterates in states 2 and 3 till the given condition is satisfied. The square operation is always executed but the computation of steps 5 from algorithm 2 depends on e[1] on line 3. As a result, a 2-by-1 multiplexer chooses between P and Q. The final result of the system design goes to the output via port S after the post-processing step.

3.1.1. Modular Multiplication Design

We propose a hardware design of Montgomery multiplication algorithm for low area complexity. The Montgomery algorithm suggested as algorithm 7 is a slightly modified copy of algorithm 4. From algorithm 4 line 4, computation of S[i+1] depends directly on the pre-computation of qi which evaluates to a zero(0) or one(1). Estimation of odd or even numbers (line 3) can straightforwardly be derived from the least significant bit where 0 equals even and 1 means odd. Deduction can be made that qi is influenced by X[i]. When X[i] equals 0, qi depends on S[0], else it depends on the XOR of S[0]and Y[0]. On the basis of this observation, a new algorithm was proposed (algorithm 7). The new algorithm splits qi into two: q_0 and q_1. Whereas q_0 computes the equivalent qi when X[i] is 0, q_1 operates when X[i] is 1. A logic circuit of bit operations can model the given mathematical equations as Q_logic. Other momentous changes that contributed to the area optimization of the new multiplier design is the elimination of the barrel register full adder and no inclusion of extra circuits such as lookahead or bypass.

Algorithm 7: Montgomery Multiplication (Proposed)
Input: X, Y, N
Output: (S1[n+2],S2[n+2])
1. Initialize S1[0] = S2[0] = 0;
2. for i = 0 to n + 1 {
3. q_0 = (S1[i] + S2[i]);
   q_1 = (S1[i] + S2[i] + Y0);
4. if (Xi = q_0 = 0)
5. (S1[i+1],S2[i+1]) = (S1[i] + S2[i]) / 2;
6. else if (Xi = 0 and q_0 = 1)
7. (S1[i+1],S2[i+1]) = (S1[i] + S2[i] + N) / 2;
8. else if (Xi = 1 and q_1 = 0)
9. (S1[i+1],S2[i+1]) = (S1[i] + S2[i] + Y) / 2;
10. else if (Xi = q_1 = 1)
11. (S1[i+1],S2[i+1]) = (S1[i] + S2[i] + Y + N) / 2; }
12. return (S1[n + 2], S2[n + 2]);

Figure 4 illustrates the structural diagram of the proposed multiplier. Register X, Y and N store the inputs and S1 and S2 accumulate the intermediate computations until the process is complete. Based on the control signals to the multiplexers, different computations are performed. When X[i] = 0, q_0 from the Q_logic is actively involved otherwise q_1. The arithmetic unit was designed as a two-level adder. Note that S1 and S2 are shift registers that outputs a 1-bit right shift. All lines in short dashes represent bit signals.

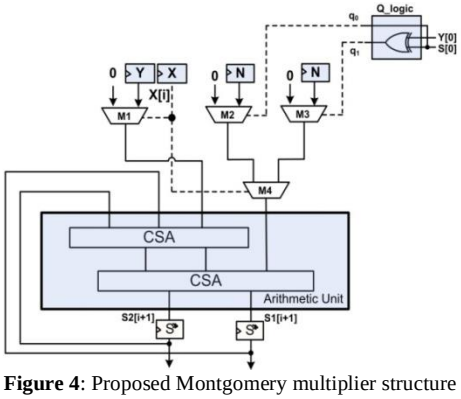

Figure 4: Proposed Montgomery multiplier structure

3.2. Pseudorandom Number Generation Design

The Akari_1, a pseudo-random number generator for secure lightweight systems, was employed. Few modifications were made to the original scheme in algorithm 5. This module takes in as input an initial seed. After 63 iterations, a random number is produced. To ensure large odd numbers are generated, the final output mapping was altered. The LSB of all numbers generated are stuck to 1 to produce only odd numbers which are pre-request for prime number except 2. In addition, the first byte i.e. the MSB is hardwired to 1. Since RSA requires large prime numbers, the modification serves the purpose right. After the initial seed, subsequent once are computed at the end of the 63rd iteration. Figure 5 illustrates the modified output mapping.

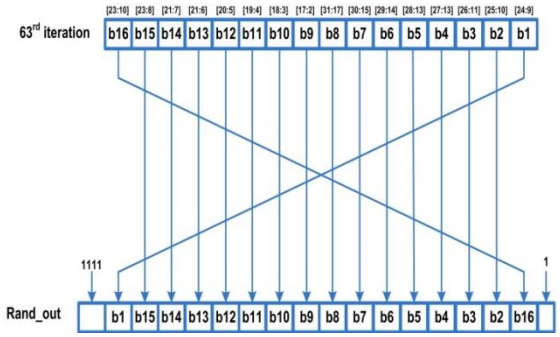

Figure 5: Output mapping (Akari_1 modified)

3.3. Primality Testing Design

Algorithm 6 was used as the backbone for testing prime numbers in the module/design. The prime number tester from I and II required the computation of modular arithmetic. Therefore the proposed modular exponentiation for encryption was called to manage resources. The probability of a given odd number being a prime decrease as the number sequence increases. Therefore, testing for a large prime number from a random number generator without a regulatory scheme is inefficient. One of such schemes is to increment the given odd number by 2 repeatedly after a test fails until a prime number is reached. This scheme, however, is limited as the difference between two large primes could be large. A simplified yet efficient way is proposed which combines the classical randomized input and the incremental model. Figure 6 shows the random test approach. This module validates a prime number. In algorithm 8, odd_no equals the random input, P_no represents the prime number, cnt_value tracks increment and the MR( ) function checks for primality as described.

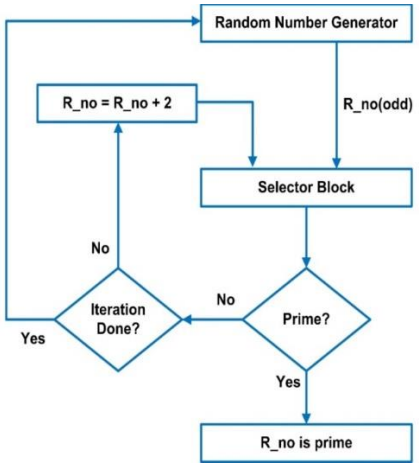

Figure 6: An efficient large prime number generation scheme

Algorithm 8: Proposed random test scheme
Input: odd_no
Output: P_no

```
1. i = 0;  cnt_value = 0;  R_no = odd_no;
2.    while(i < 500) {
          x = MR(R_no) ;
          if(x = 1)
             return (P_no = R_no + cnt_value);
          else {
                cnt_value = cnt_value + 2;
                R_no = R_no + cnt_value;  }
          i++;  }
3. Go back to step 1
```

3.4. Key Generation Design

The key generator model employs the generic RSA key generation algorithm as described in section 2. Given two distinct numbers p and q, the key generator produces the public exponent, private exponent and modulus. The generator makes use of the greatest common divisor (GCD) module built from the binary Euclidean algorithm. To reduce the computational complexity in finding GCD, all inputs are made odd by consecutively removing zeros or performing a right shift in binary implementation. The GCD calculation aids in determining the public exponent e which should be relatively prime to the totient. Evaluation of the d is made possible by finding the multiplicative inverse of $e \bmod \emptyset(n)$. The Extended Euclidean Algorithm (EEA) is used for computing modulo inverse [18, 19].

4. Results and discussion

Synthesis of the proposed hardware design was done using the TMSC 90nm and 130nm CMOS technology from Synopsys for the Montgomery multiplier and modular exponentiation core respectively. Table 1 shows comparisons of different multiplier designs. The symbol * in Table 1 denotes the worst-case scenario or the maximum number of clock cycles for one Montgomery multiplication. Our proposed Montgomery multiplier experienced a significant reduction of about 47.19% in area compared to paper [13] at 250MHz. This reduction is due to simplified Q_logic using bit operations, the omission of the BRFA with no special circuitry such as lookahead tables or bypass logic and compact design of the arithmetic unit. Furthermore, a clock speed of 1.5ns was gained. Nevertheless, [13] recorded fewer clock cycles compared with our approach at 4ns, hence better throughput.

Table 1: Result comparison of different Montgomery multiplier design for 1024-bit key size

Montgomery Multiplier	Process	Cycles	Delay (ns)	Area (um^2)	Throughput(Mbps)
Proposed_1	90nm	1026*	1.50	332K	665.4
Proposed_2	90nm	1026*	4.00	263K	249.5
[6]	90nm	1049	5.60	406K	174.3
[13]	90nm	880	4.00	498K	290.9

Table 2 illustrates implementation comparisons for 1024-bit exponentiation units. An operating frequency of 452.49MHz was used to establish a common platform for comparison with reference paper [7]. The number of gate count of our modular exponentiation unit decreased approximately 28.18%.

Table 2: Implementation comparison for 1024-bit exponentiations

Modular Exponentiation	Process	Delay (ns)	Area (um^2)	Gate Count (K)
Proposed	130nm	2.21	535397	79
[7]	130nm	2.21	714676	110
[17]	130nm	2.00	-	139

5. Conclusion

In this paper, an RSA cryptosystem was designed with a focus on hardware area. An area-efficient Montgomery multiplier was proposed in this study. The proposed hardware design eliminated some components of the existing design and presented a simplified Q_logic. The Q_logic complimented with the compact arithmetic unit caused the hardware area to decrease. The arithmetic unit employed a two-level Carry Save Adder which is optimal for three operand addition. The new Montgomery multiplier was embedded into a modular exponentiation unit. The multiplier block inside the exponentiation module was designed to be used once per cycle sacrificing speed for area. The key generator core, given two distinct prime numbers from the PRNG module and validated by the primality testing module, generates the public-private key pair. The keys are used for encryption and decryption. Resource sharing design was a major factor in the proposed cryptosystem.

Operating frequencies of 666.67MHz and 452.49MHz were achieved for the Montgomery multiplier and modular exponentiation respectively with corresponding gate count of 60K and 79K. It was observed from the research that the divider module influenced the system speed. Designing a faster and low area divider will improve the system drastically.

Acknowledgment

This research was supported by the MSI (Ministry of Science, ICT and Future Planning), Korea, under the Global IT Talent support program (IITP-2017-0-01681) supervised by the IITP (Institute for Information and Communication Technology Promotion)

References

[1] Nayak SK, Mohanty S, Majhi B (2017), CLB-ECC: Certificateless blind signature using ECC. *J. Information Processing Systs.* 13, 970–986.

[2] Rivest L, Shamir A, Adleman L (1978), A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM* 21, 120–126.

[3] Koblitz N (1987), Elliptic curve cryptosystems. *Math. Comput.* 48, 203–209.

[4] Montgomery PL (1985), Modular multiplication without trial division. *Math. Comput.* 44, 519–521.

[5] Kim YS, Kang WS, Choi JR (2000), Asynchronous implementation of 1024-bit modular processor for RSA cryptosystem. *Proc. 2nd IEEE Asia-Pacific Conf. ASIC*, 187–190.

[6] Zhang YY, Li Z, Yang L, Zhang SW. (2007), An efficient CSA architecture for Montgomery modular multiplication. *Microprocessors Microsyst.* 31, 456–459.

[7] Kuang SR, Wang JP, Chang KC, Hsu HW (2013), Energy-efficient high-throughput Montgomery modular multipliers for RSA cryptosystems. *IEEE Trans. Very Large Scale Integration (VLSI) System* 21, 1999–2009

[8] McIvor C, McLoone M, McCanny JV (2004), Modified Montgomery modular multiplication and RSA exponentiation techniques. *IEE Proc. Comput. Digit. Techn.* 151, 402–408

[9] Koç ÇK (1994), High-speed RSA implementation. Technical Report, RSA Laboratories, RSA Data Security Inc., 46–49.

[10] Daly A, Marane W (2002), Efficient architectures for implementing Montgomery modular multiplication ad RSA modular exponentiation on reconfigurable logic. *Proc. of the 2002 ACM/SlGDA 10th international symposium on FPGA ACM*, 40–49

[11] Walter CD (1999), Montgomery exponentiation needs no final subtractions. *Electronics Letters Journal* 35, 1831–1832

[12] Hu ZB, Shboul RMA, Shirochin VP (2007), An efficient architecture of 1024-bits cryptoprocessor for RSA cryptosystem based on modified Montgomery's algorithm. *Proc. 4th IEEE Int. Workshop Intell Data Acquisit. Adv. Comput. System*, 643–646

[13] Kuang SR, Wu KY, Lu RY (2016), Low-cost high-performance VLSI architecture for Montgomery modular multiplication. *IEEE*

Trans. Very Large Scale Integration (VLSI) System 24, 440-442.
[14] Martín H, Millán ES, Entrena L, Castro JCH, López PP (2011), AKARI-X: a pseudorandom number generator for secure lightweight systems. *IEEE 17th Conf. On-Line Testing Symposium.*
[15] Miller GL (1976), Riemann's hypothesis and tests for primality. *J. Comput. Syst. Sci.* 13, 300–317.
[16] Rabin MO (1980), Probabilistic algorithm for testing primality. *J. Number Theory* 12, 128–138.
[17] Shieh MD, Chen JH, Wu HH, Lin WC (2008), A new modular exponentiation architecture for efficient design of RSA cryptosystem. *IEEE Trans.Very Large Scale Integration (VLSI) System* 16, 1151–1161.
[18] Mallick PK, Kamila NK (2011), Crypto steganography using linear algebraic equation. *Int. J. Comput. Comm. Techn.* 2, 106–112.
[19] Mallick PK, Kamila NK, Patnaik S (2011), Computing symmetric block cipher using linear algebraic equation. *Int. J. Comm. Net. Sec.* 1, 7–11.