An Efficient Montgomery Multiplication Algorithm and RSA Cryptographic Processor

Richa Garg
University Institute of Engineering & Technology, Sector-25, Panjab University, Chandigarh. India
E-mail- richa.garg1@gmail.com

Renu Vig
University Institute of Engineering & Technology, Sector-25, Panjab University, Chandigarh. India
E-mail- renuvig@hotmail.com

Abstract

New, generic silicon architecture for implementing Montgomery's multiplication algorithm is presented. This paper proposes an efficient Montgomery modular multiplication technique that employs multi-bit shifting and carry-save addition to perform long-integer arithmetic. The gain in data throughput for Montgomery multiplication is approximately 45.49% (for 1024-bit length) and the hardware reduction is 24.27% of the traditional methods. Hence, the corresponding hardware realization is optimal in terms of area and offer higher data throughput for Montgomery multiplication. The practical application of this approach has been demonstrated by applying this to the design of RSA processor architecture with 512-bit and 1024-bit key size. The RSA processor also offers higher throughput (32.18% for 1024-bit) with a slight increase in area (22.4%). This optimization is also technology independent and thus should suit well not only FPGA implementation but also ASIC. The Montgomery design and RSA processor has been evaluated on Xilinx Virtex-4series for the practical bit lengths of 512, 1024 and 2048. The resulting Montgomery multiplier and the RSA processor performance results presented ate the fastest reported to date in literature.

I. Introduction

The Montgomery multiplication methods constitute the core of the modular exponentiation operation, which is the most popular method in public-key cryptography for encrypting and signing digital data [1]. It is a time-consuming arithmetic operation because it involves multiplication as well as division [2]. For security reasons, the bit length of the operands are generally in the range of 512-2048 bits for RSA and achieving high throughput rates for these bit precisions is a challenging task. Speeding the modular multiplication will have a great impact on the speed of Public-key algorithms like RSA and ECC which are heavily used in current cryptographic systems.

Among the different ways of performing modular multiplication, Montgomery modular multiplication [4] introduced by Peter Montgomery is the widely used algorithm. It replaces the trial divisions by modulus with a series of additions and divisions by two suitable for VLSI implementations [4-5]. There are several techniques proposed in literature that avoid the carry propagation involved in addition circuitry of Montgomery algorithm which is a key factor in determining the overall performance [6-9]. These include breaking up of addition into multiple stages to best utilize the dedicated carry chains in FPGAs [6] and use of FPGA systolic array multiplier architectures with varying processing element sizes [7]. Bunimov et.al [8] proposed a technique that uses carry-save logic to reduce the critical path delay. However, due to different input/output formats used, the design needs conversion from output to input format in applications like RSA where repeated modular multiplications are

0-7695-3050-8/07 $25.00 © 2007 IEEE
DOI 10.1109/ICCIMA.2007.272

needed. Recently McIvor et.al [3] proposed two algorithms that avoid the conversions required in [8]. The algorithms are suitable for RSA exponentiation and have been proved to be better than the ones existing in literature [6-8]. The algorithms use carry-save arithmetic on operands to avoid the large propagation delays involved in additions and use five-to-two CSA (Carry Save Addition) and four-to-two CSA (Carry Save Addition) that use three levels and two levels of CSL (Carry Save Logic) respectively. The algorithms in [3] need k+1 and k+2 clock cycles to perform the multiplication operation using five-to-two CSA and four-to-two CSA respectively, where k is the number of bits used to represent the modulus. The authors reported that the Montgomery multiplication techniques presented were of high-throughput rates compared to any other design known in the literature. Manochehri et.al [11] proposed a Montgomery multiplication algorithm using pipelined carry save addition and has been compared with algorithms in [3]. It has been shown that the pipelined implementation is well suited for FPGAs compared to ASICs. However, the throughput of the design is less than the throughput of the design proposed in [3], but with a reduction in area. In this paper, the main focus is on improving the throughput of the design and the authors propose a modified Montgomery multiplication algorithm that has less computational delay and high throughput compared to five-to-two CSA and four-to-two CSA Montgomery multiplications [3]. The proposed design uses carry-save arithmetic and has been evaluated for 512, 1024 and 2048 operand bit lengths.

II. Montgomery Modular Multiplication

Montgomery image or residue of number a< n where 'n' is a 'k'-bit modulus is defined as $A = a.2^k \pmod n$. Given two residue numbers A, B, the Montgomery product in GF(n) is defined as k MonPro(A,B) $A*B*2^{-k} \pmod n$.

Algorithm I is the algorithms proposed in [3] based on four-to-two CSA respectively and are shown here for convenience. The inputs to the algorithms are A1, A2, B1, B2, n where (A1, A2) and (B1, B2) are the carry save representations of A, B respectively and 'n' is the modulus. In this paper, a number Z is represented as (X, Y) such that Z=X+Y. In algorithm I, A_i is calculated from (A1, A2) in parallel with CSA to avoid extra clock cycles [3].

Critical delay of Algorithm I is 2 Full Adders + 4:1 Multiplexer + 2 XOR + 1 AND = 6 XOR + 4:1 MUX + 1 AND (two full adder critical delay= 4 XOR due to four-to-two CSA that has two levels of CSL, 4:1 MUX delay to check for either of the four conditions determined by A_i and q_i, 2 XOR+1 AND to compute q_i).

A. Algorithm I. *Four-to-two CSA Montgomery Multiplication (A1, A2, B1, B2, n) [3]*

D1,D2 = CSR (B1 + B2 + n + 0)
S1[0] = 0;
S2[0] = 0;
for i in 0 to k-1 loop
    $q_i = (S1[i]_0 + S2[i]_0) + (A_i * (B1_0 + B2_0)) \bmod 2$;
if $A_i = 0$ and $q_i = 0$ then S1[i+1], S2[i+1] = CSR (S1[i] + S2[i] + 0 + 0) div 2;
if $A_i = 0$ and $q_i = 0$ then S1[i+1], S2[i+1] =CSR (S1[i] + S2[i] + 0 + 0) div 2;
if $A_i = 0$ and $q_i = 0$ then S1[i+1], S2[i+1] =CSR (S1[i] + S2[i] + 0 + 0) div 2;
if $A_i = 0$ and $q_i = 0$ then S1[i+1], S2[i+1] =CSR (S1[i] + S2[i] + 0 + 0) div 2;
end loop;

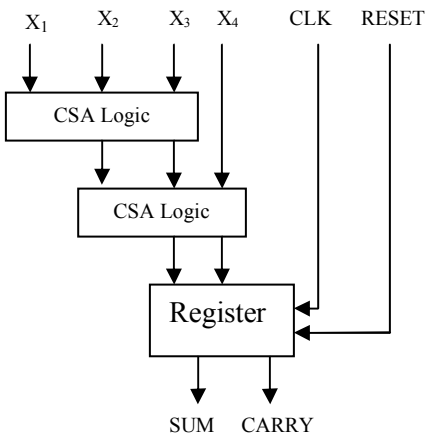

Figure 1 Block Diagram of Four-to-Two CSA

III. Algorithm II

Algorithm II [9] illustrates a new technique to perform Montgomery modular multiplication. The inputs to the algorithms are A1, A2, B1, B2, n where (A1, A2) and (B1, B2) are the Carry save representations of A, B respectively and 'n' is the modulus. The algorithm uses four-to-two and three-to-two carry save arithmetic to avoid delay due to carry propagation involved in additions. In algorithm II, 2-bit shifting where least significant two bits of 'A' are processed instead of processing one bit at a time is introduced. Here, $\{a_{2i+1}, a_{2i}\}$ are computed using the component shown in figure 2, where { } indicates concatenation of bits. $\{a_{2i+1}, a_{2i}\}$ is calculated by adding the two least significant bits of A1 and A2 using a 2-bit adder. After this computation has been performed, carry-out of the adder is fed as carry-in for the next iteration and the values in A1, A2 are barrel shifted by two places to the right so that the next set of values can be computed. These values are being computed in parallel with the carry save adders and hence no extra clock cycles are needed. To the author's knowledge, the area utilization is less as compared to the existing Montgomery algorithms.

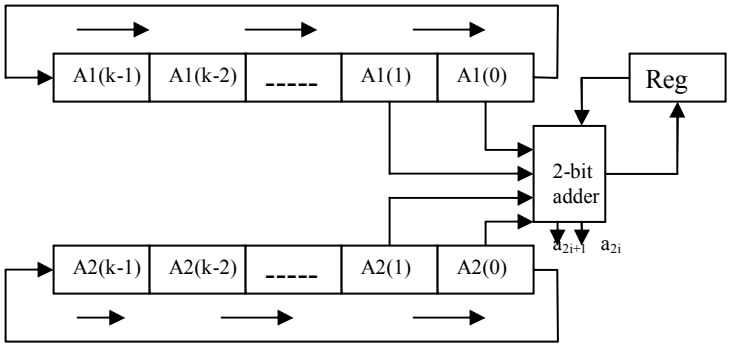

Figure 2 Computation of a_{2i+1} and a_{2i} from A1 , A2

B. Algorithm II.*(Modified Montgomery multiplication using Carry-Save arithmetic) [9]*
Montgomery Multiplication (A1, A2, B1, B2, n)
1. 1a. (S1[0], S2[0]) = (0, 0)
   1b. (B1_ 3, B2 _ 3) = CSR ({B1, 0} + B1+ {B2, 0} + B2) and
   N3 = 3 *n (precomputed)

2. For i in 0 to [k/2]-1 loop

2a. if ($\{a_{2i+1}, a_{2i}\}$ = = 00) then BB1= 0; BB2 = 0;
else if = ($\{a_{2i+1}, a_{2i}\}$ = = 10) then BB1= B1; BB2 = B2;
else if = ($\{a_{2i+1}, a_{2i}\}$ = = 01) then BB1= {B1, 0}; BB2 = {B2, 0};
else BB1= B1_3; BB2 = B2_3;

2b. S1[i+1], S2[i+1]) = CSR (S1[i] + S2[i] + BB1 + BB2);

2c. $\{q_{2i+1}, q_{2i}\} = (\{S1[i+1]_1 \text{ xor } S2[i+1]_1\}, S2[i+1]_0)$;

2d. if ($\{q_{2i+1}, q_{2i}\}$ == 00) then N = 0;
else if ($\{q_{2i+1}, q_{2i}\}$ == 01) then N = n;
else if ($\{q_{2i+1}, q_{2i}\}$ == 10) then N = {n, 0};
else N = N3;

2e. S1[i +1],S2[i+ 1]) = CSR (S1[i +1] + S2[i+ 1] + N) ;

2f. if ($(S1[i+1]_1 * S2[i+1]_1) == 1$) then
$(S1[i+1],S2[i+1]) = CSR\ (S1[i+1]_{k-1:2} + S2[i\ 1]_{k-1:2} + 1)$;
else $(S1[i+1],S2[i+1]) = CSR\ (S1[i+1]_{k-1:2} + S2[i\ 1]_{k-1:2})$;
end loop;

3. Return (S1[k/ 2],S2[k / 2]);

Critical delay analysis for Algorithm II:
Step 2a: 4:1 MUX delay to calculate BB1, BB2
Step 2b: 4 XOR delay due to four-to-two CSA
Step 2c: 1 XOR delay for calculation of $\{q_{2i+1}, q_{2i}\}$
Step 2d: 4:1 MUX delay to calculate N
Step 2e: 1 Full adder i.e., 2 XOR delay due to three-to- two CSA
Step 2f: 1 AND delay to check the condition
Else loop selects (k-2) most significant bits and does not add to critical delay.
Therefore, the critical delay adds up to 4:1 MUX + 4 XOR + 1 XOR + 4:1 MUX + 2 XOR + 1 AND + 2 XOR = 9 XOR + 2 4:1 MUX + 1 AND.

The algorithm II requires K+1 clock cycles to compute the Montgomery modular product. Estimation of latency in calculation of Montgomery modular multiplication using algorithms I and II are as follows: (Excluding delay due to routing)
Let X= XOR delay, A=AND gate delay and M=4:1MUX delay
Algorithm I: (k+2)*(6X+M+A) + 4X = (6k+16)X + (k+2)M + (k+2)A
Algorithm II: (k+1)*(9X+2M+A) + 4X = (9k + 9)X + (k+2)2M + (k+1)A

A delay of 4 XOR is added to the total delay for algorithm I, II due to pre-computation that uses a four-to-two CSA. In algorithm Ii, (3*n) is pre-computed and since the modulus is fixed for a sequence of operations as in RSA exponentiation, this does not add up to the delay calculation of the modular multiplication. From these, it is inferred that delay due to algorithm II reduces drastically. Technology independent optimization is quite obvious and the proposed algorithm well suits for efficient implementation on both ASIC and FPGA.

The functionality of proposed design is captured generically in VHDL and implemented into the Xilinx Virtex4 series of FPGAs for varying bit lengths. Table I gives performance results for these implementations. These results can be directly compared with those given in [3] for the four-to-two CSA multiplier. Fig. 3 to 8 provides a graphical representation of the comparisons in terms of percentage increase in data throughput rate and area when using the three-to-two and four-to-two CSA architecture rather than only four-to-two design.

The next section will demonstrate the practicality of the multiplier architectures through an implementation of the RSA cryptosystem for 512-bit and 1024-bit key sizes. For these key sizes, the proposed CSA multiplier is used to calculate the successive modular multiplications.

Table I
Comparison Of Frequency, Area And Throughput For Montgomery Algorithm 1 And 2

	Bit Length & FPGA used	Frequency (MHz)	Area (slices)	Throughput (Mbps)
Algo1	512(XC2V1500)	122.03	5782	121.55
	1024(XC2V3000)	111.32	11520	111.10
	2048(XC2V6000)	90.73	23108	90.64
Algo2	512(XC4VSX35)	179.746	4187	179.746
	1024(XC4VSX35)	161.645	8724	161.645
	2048(XC4VSX55)	131.032	17520	131.032

IV. RSA Processor Architecture

C . The RSA Crypto processors Architectures

The RSA encryption and decryption functions are given by $C=M^e(mod\ n)$ and $M=C^d(mod\ n)$ respectively, where M is a plaintext message block, C is a ciphertext block, n is the k-bit modulus, and e and d are the public and private exponents respectively. The equation $ed=1((mod(p-1)(q-1))$ must also hold, where p and q are two large prime numbers (~ $k/2$-bits in length) and $n=pq$.

Thus, an RSA operation is a modular exponentiation with operands satisfying the conditions stated above.

By using the proposed CSA multiplier architecture to calculate the MontMult (Montgomery Multiplication) stages, we can re-write this algorithm as Algorithm III, given below. As a result, we have been able to develop RSA processor architectures with very impressive data throughput rates.

Algorithm III. Modified Modular Multiplication (C, d, n) [3]
K = 2^{2k} (mod n); (computed externally)
P1[0], P2[0] = modified_MontMult (K, 0, C, 0, n);
R1[0], R2[0] = modified_MontMult (K, 0, 1, 0, n);
for i in 0 to d_k loop
P1[i+1], P2[i+1] = modified_MontMult (P1[i], P2[i], P1[i], P2[i], n);
if d[i] = 1 then R1[i+1], R2[i+1] = modified_MontMult (R1[i], R2[i], P1[i], P2[i], n)
end if;
end loop;
M1, M2 = modified_MontMult (1, 0, R1[k], R2[k], n);
M = M1 + M2;
return M;

RSA processor architectures based on the proposed CSA multiplier described were captured generically in VHDL and implemented using the Xilinx Virtex4 series of FPGAs (using the highest place and route effort level) for 512-bit and 1024-bit key sizes. Tables II

provide performance comparison for these implementations. A public exponent of $2^{16}+1$ and a *k*-bit private exponent were used during testing.

Table II
Comparison Of Frequency, Area And Throughput For RSA Processor

	Bit Length & FPGA used	Frequency (MHz)	Area (slices)	Throughput (Mbps)
Algo1	512(XC2V3000)	96.27	11,196	4.79
	1024(XC2V6000)	93.34	22,075	4.66
Algo2	512(XC4VSX35)	138.293	13,676	6.90
	1024(XC4VSX35)	123.211	28,891	6.16

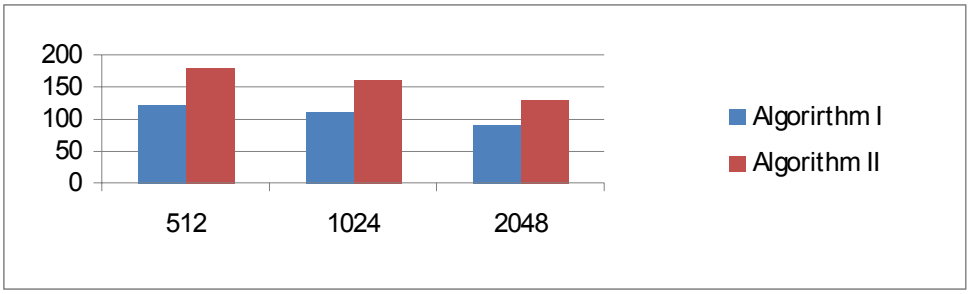

Figure 3 Comparison of Frequency for Algorithm I and II

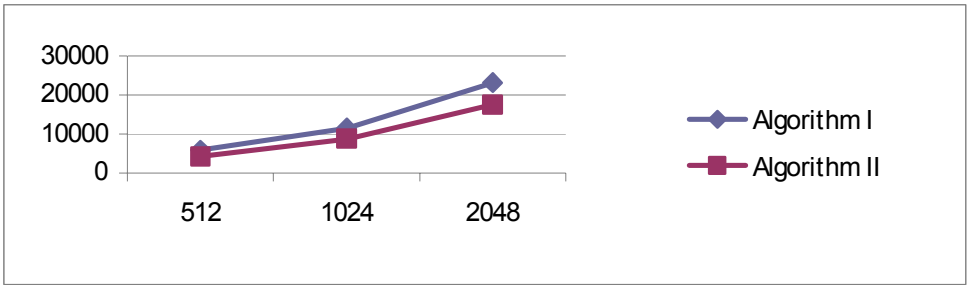

Figure 4 Comparison of area (in terms of slices on FPGA) of Algorithm I and II

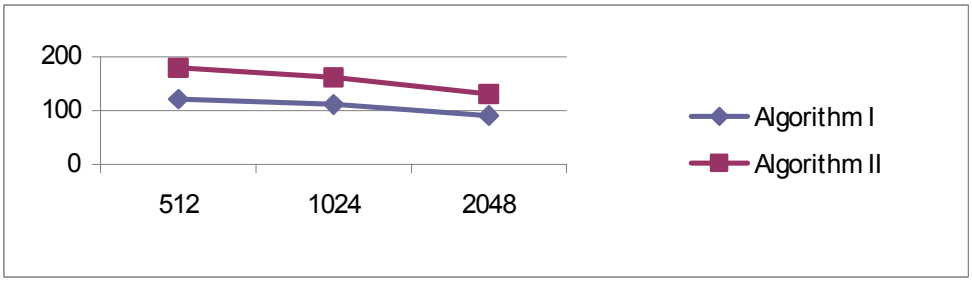

Figure 5 Comparison of Throughput of Algorithm I and II

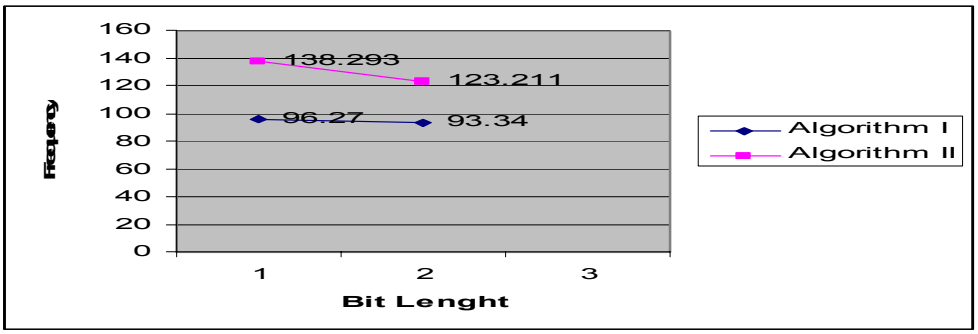

Figure 6 Comparison of Frequency for RSA processor

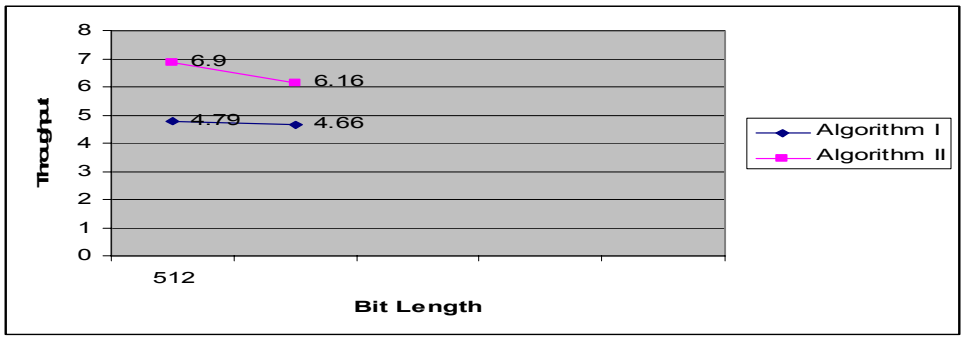

Figure 7 Comparison of Throughput for RSA processor

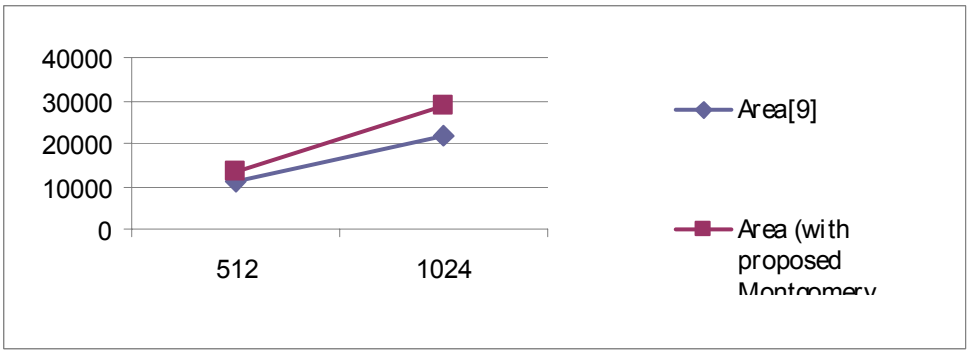

Figure 8 Comparison of Area for RSA processor

IV. Results and Discussions

The functionality of the proposed algorithm has been verified by capturing the silicon based design using VHDL and targeted to Virtex4 Series FPGA for varying operand lengths of 512, 1024 and 2048 bit for Montgomery Algorithm and for RSA Processor. The results obtained for the proposed algorithm and the results provided in [3] for algorithms I is shown in Table I and for RSA processor, results are provided in Table II. The experimental results are very close to the expected values produced by the theoretical expressions. The minor differences are due to the routing, buffer delays introduced when targeted to FPGA. Both algorithms have been evaluated for 512, 1024 and 2048 bit lengths which are of practical interest in cryptographic applications. Hence, it has been shown that by using two bits of A while addition using BRFA component, the area can be reduced and the higher speed can be

achieved. The resulting Montgomery multiplier results presented are, we believe, the fastest reported in the literature to date.

V. Conclusion

In this paper, modified Montgomery modular multiplications that employs multi-bit shifting and carry save arithmetic to avoid carry propagation problems has been implemented. The main contribution of the paper is in using multi-bit shifting and modifying the Montgomery modular multiplication algorithms available in literature to achieve high data throughput rate. The practical application of this approach has been demonstrated by applying this to the design of RSA processor architecture with 512-bit and 1024-bit key size. The functionality of the RSA algorithm has been verified by capturing the silicon based design using VHDL and targeted to Virtex-2 Pro and Virtex-4 Series FPGA for varying operand lengths of 512, 1024 and 2048 bit. The proposed method is a technology independent optimization and suits both ASIC and FPGA implementations. The design has been targeted to Xilinx Virtex4 series FPGA and tested for practical lengths of 512, 1024 and 2048 bits. The increased data throughput rate for the proposed algorithm is expected to play an eminent role in high-speed cryptographic applications.

Acknowledgments

The authors would like to thank Government of India, Ministry of Communications and Information Technology, Department of Information Technology, New Delhi, for funding the Project "Development of Software Protection Tools", under which this work has been done.

References

[1] William Stallings, "*Cryptography and Network Security, Principles and practices*" Edition 3d, Pearson Education.

[2] A. Menezes, P.Van Oorschot, and S.Vanstone, "*Handbook of applied Cryptography*", CRC Press, 1996.

[3] C.McIvor, M.McLoone and J.V.McCanny, "Modified Montgomery modular multiplication and RSA exponentiation techniques", *IEE Proc.-Comput. Digit. Tech*., November 2004, Vol. 151, No. 6, pp. 402-408.

[4] Montgomery P.L, "Modular Multiplication without trial division", *Math. Comput, 1985*, Vol 44, No.70, pp.519-521.

[5] Eldridge S.E, Walter C.D, "Hardware Implementation of Montgomery's Modular Multiplication Algorithm", *IEEE Transactions,* 1993, Vol. 42, pp.693-699.

[6] Elbirt A.J, Paar.C, "Towards an FPGA Architecture optimized for Public-key algorithms", *Proceedings of SPIE Symposium on Voice, Video and Communications*, Sept. 1999

[7] Blum.T, Paar.C, "Montgomery modular exponentiation on Reconfigurable Hardware", *Proceedings of 14th Symposium on Computer Arithmetic,* 1999, pp.70-77

[8] Bunimov.V, Schimmler.M, Tolg.B, "A Complexity-Effective version on Montgomery's Algorithm", *Proceedings of Workshop on Complexity Effective Designs (WECD 2002),* May 2002.

[9] Kamala RV, Shrinivas MB, "High-Throughput Montgomery Modular Multiplication", *proceedings IEEE,* 3-901882-19-7 2006 IFIP, 2006

[10] Cilardo.A, Mazzeo.A, Romano.L, Saggese.G.P, "Carry-Save Montgomery Modular Exponentiation on Reconfigurable Hardware", *Proceedings of Design, Automation and Test in Europe (DATE),* 2004, vol.3, pp. 206-211.

[11] Kooroush Manochehri, Saadat Pourmozafari, "Fast Montgomery Modular Multiplication by Pipelined CSA Architecture", *Proceedings of 16th International Conference on Microelectronics,* ICM2004, pp.144-147.

[12] C.K. Koc, T. Acar, and B. Kaliski (June 1996) Analyzing and Comparing Montgomery Multiplication Algorithms, *IEEE Micro,* vol. 16, no. 3, pp. 26-33.