ISSN 2047-3338

FPGA Implementation of Modified Montgomery for RSA Cryptosystem

Rupali Verma[1], Maitreyee Dutta[2] and Renu Vig[3]
[1]PEC University of Technology, Chandigarh, India
[2]National Institute of Technical Teachers Training and Research, Chandigarh, India
[3]University Institute of Engineering and Technology, Panjab University, Chandigarh, India
[1]rupali@pec.ac.in

***Abstract*– Modular multiplication is a key operation in public key cryptosystems like RSA. Among modular multiplication methods, Montgomery modular multiplication is an efficient algorithm suitable for hardware implementation. In this paper, a Modified Montgomery Modular Multiplication design is proposed with carry save adder architecture and parallel simplified quotient computation for the next iteration. The proposed design has a high clock frequency and high throughput. The proposed design and RSA are implemented on Virtex 2 and Virtex 5 FPGAs.**

***Index Terms*– Carry Save Adders, Cryptography, Modular Multiplication and RSA Exponentiation**

I. INTRODUCTION

RSA is a popular public key cryptosystem for encryption and digital signatures [1]. Encryption, decryption and digital signatures in RSA are function of modular exponentiation which is achieved by repeated modular multiplications. In 1985, P.L. Montgomery [2] proposed an efficient method for modular multiplication. It is suitable when several computations are done modulo one n (modulus). RSA cryptosystem requires repeated modular multiplications using the same modulus. For security reasons, the size of operands in RSA cryptosystem is 1024 bits or more. As a result the critical operation in Montgomery modular multiplication is addition of large operands. To avoid carry propagation during addition, several architectures are proposed in literature such as systolic array modular multipliers [3, 4] and carry save adder architectures [6-11]. This paper focuses on implementations based on carry save addition (CSA). Many researchers have proposed modifications in the design of Montgomery modular multiplication to achieve high frequency. Mc Ivor et.al [6, 7] proposed two Montgomery modular multiplication architectures: five-to-two CSA (three levels of carry save logic) and four-to-two CSA with two additional registers (two levels of carry save logic). The two variants perform modular multiplication in k+1 and k+2 (k is operand length in bits) clock cycles respectively. Kooroush Manochehri et.al [8] has proposed Montgomery modular multiplication algorithm using pipelining and carry save adder architecture. The authors have compared their result with [6, 7] and have found that pipelining is useful for FPGA implementation. The New Montgomery multiplication proposed by Kooroush Manochehri et.al [9] has higher throughput than [6, 7] as it calculates quotient in parallel with the addition of operands. Ming-Der Shieh [10] proposed new modular exponentiation architecture with unified multiplication/square module in which the number of operands was reduced by mathematical manipulation. But the design in [10] could be used only for H algorithm (MSB first method) of modular exponentiation. The critical path in 4:2 CSA [6, 7] consists of computation of quotient and 2 levels of CSA. The quotient determination involves delay of 2 XOR and 1 AND. Therefore the critical path delay is 2 full adders + 4:1 multiplexer + 2 XORs + 1 AND. This design has a large delay in quotient computation which is solved in our previous work [11]. The design in this paper further reduces the critical path of Montgomery design by computing quotient for next iteration in parallel with carry save addition thus, saving the delay of 2 XOR and 1 AND. Hence the critical path of our design reduces to 4:1 MUX+2 Full adders.

II. MONTGOMERY MODULAR MULTIPLICATION

Montgomery modular multiplication (MMM) is an efficient method since it replaces trial division by modulus with additions and shift operations. These operations are easily performed on hardware. But the price paid in using Montgomery algorithm is conversion of operands in and out of Montgomery's domain. Let a and b be the multiplier and multiplicand, n be the modulus. To compute Montgomery modular multiplication (MMM) of a and b modulus n the operands are first converted to Montgomery domain i.e., n residue of integer with respect to r, where $r=2^k \bmod n$ and k is bit length of operands:

A is n residue of a with respect to r
$A=a*r \pmod n$
Similarly $B = b*r \pmod n$

The n residue of integer with respect to r is obtained by Montgomery modular multiplication of integer and r^2

A= MMM (a, r^2, n) = (a* r^2 * r^{-1}) mod n
= a*r (mod n)
Montgomery product of A and B is defined as
=MMM (A, B, n) = A*B* r^{-1} mod n
= (a*r (mod n)) * (b*r (mod n)) * r^{-1} mod n
Product = ab r mod n
Product is converted back to integer domain from Montgomery domain.
= MMM (Product, 1, n)
= a b r * 1 * r^{-1} mod n
= ab mod n.

In RSA cryptosystem modular exponentiation is achieved by repeated modular multiplications with the same modulus. Hence the cost of conversion of input operands a and b from integer domain to Montgomery domain and then the result back to integer domain is very negligible in cryptosystems like RSA.

Algorithm 1: MMM (A, B, n)

// Montgomery's modular multiplication algorithm
// Inputs: n (modulus k bits), A (multiplier in Montgomery's domain, k bits)
// B (multiplicand in Montgomery's domain, k bits)
// A, B < n
//Output S=A × B × r^{-1} mod n
1. S=0;
2. for i=0 to k-1
3. {q=(S+A[i]×B) mod 2;
4. S= (S+ A[i] × B +q × n)/2; }
5. if (S ≥ n)
6. S=S-n;
7. return S;

Algorithm 1 is a radix 2 version of Montgomery's multiplication algorithm [10]. The critical path in Montgomery's design is addition of long operands which results in large carry propagation delay. Algorithm 2 is 4:2 CSA [6, 7] where carry propagation is avoided by use of carry save adders.

Algorithm 2: Four-to-two CSA Montgomery Multiplication (A1, A2, B1, B2, n)

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$_0$)) mod 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;
elsif (A_i=1 and q_i =0) then
S1[i+1], S2[i+1]
= CSR (S1[i] + S2[i] + B1 + B2) div 2;
elsif (A_i=0 and q_i =1) then
S1[i+1], S2[i+1]
= CSR (S1[i] + S2[i] + n + 0) div 2;
else
S1[i+1], S2[i+1]
= CSR (S1[i] + S2[i] + D1+ D2) div 2;
end if;
end loop;
return S1[k],S2[k];

The critical path in 4:2 CSA Montgomery consists of computation of quotient and 2 levels of carry save addition. The critical path delay of algorithm 2 is 2 full adders + 4:1 multiplexer + 2 XORs + 1 AND. Our previous work [11] reduces the delay in 4:2 CSA by simplifying quotient computation. It is done by increasing the bit length of multiplicand and multiplier by one bit and computing quotient for next iteration in 1 XOR delay. In this paper we propose a Modified Montgomery Modular Multiplication design which is a hybrid of the two designs: four to two CSA proposed by McIvor [7] and design in [4]. Two bits of multiplicand are increased in [4] to make quotient for next iteration independent of partial product in the current iteration. The work in this paper reduces the critical path of 4:2 CSA Montgomery by computing quotient for next iteration in parallel to carry save addition. The quotient computation is simplified by mathematical manipulation which is discussed in next section.

III. PROPOSED MONTGOMERY MODULAR MULTIPLICATION

Algorithm 3: Proposed Montgomery Modular Multiplication (A1, A2, B1, B2, n)

1. 1a. (S1[0], S2[0]) = (0, 0)
1b. B1_n= 4B1, B2_n=4B2; // loading state parallel
1c A1_n= 00 & A1; A2_n =00 & A2; parallel
1d P1=0, P2=0;
2. 2a. D1, D2 = CSR (B1_n + B2_n + n);
2b. q_0 = 0;
2c. Computation of A_n$_0$ for first iteration // pre-computation state
3. for i in 0 to k+1 loop
3a. { if (A_n$_i$=0 and q_i =0) then P1=0 ; P2=0;
elsif (A_n$_i$=1 and q_i =0) then
P1= B1_n; P2=B2_n;
elsif (A_n$_i$=0 and q_i =1) then
P1= 0; P2=n;
else P1= D1; P2=D2;
end if;
3b S1[i+1], S2[i+1]
= CSR (S1[i] + S2[i] + P1 + P2) div 2; }
(steps 3a to 3b parallel to step 3c)
3c. if (q_i =0) then
q_{i+1}=(S1[i]$_1$ ⊕S2[i]$_1$) ⊕ (S1[i]$_0$·S2[i]$_0$);
else
q_{i+1} =(S1[i]$_1$ ⊕ S2[i]$_1$ ⊕ n_1 ⊕ 1);
end if; // computation of quotient
// n_1 the second least significant bit of n
(steps 3a to 3b parallel to step 3d)
3d. { computation of A_n $_{i+1}$ }
end loop;
4. Return (S1[k+2],S2[k+2]);

B1 and B2, the carry save representation of multiplicand becomes B1_n and B2_n in our design. The two LSB of B1_n and B2_n are 00. Hence quotient for next iteration q_{i+1} becomes independent of current partial product A_n_i (B1_n), A_n_i (B2_n) but depends on quotient for the current iteration.

Quotient for next iteration can be computed by

$$q_{i+1}= ((S1[i]_{1:0} + S2[i]_{1:0} + q_i * n_{1:0}) / 2) \bmod 2; \quad (1)$$

it can also be written as

$$q_{i+1}=((S1[i]_1+S2[i]_1+q_i*n_1) + carry(S1[i]_0+S2[i]_0+q_i*n_0)) \bmod 2; \quad (2)$$

But in 4:2 CSA when $A\_n_i=1$ and $q_i=1$ then computation of quotient for next iteration becomes

$$q_{i+1}=((S1[i]_{1:0}+ S2[i]_{1:0} + P1_{1:0}+ P2_{1:0})/2) \bmod 2; \quad (3)$$

and is only possible after determination of P1 and P2. So the work in this paper tries to simplify this quotient computation and remove the dependency on P1 and P2.

There are two cases

1) $q_i=0$

When $q_i=0$ then for both values of A_n_i, q_{i+1} become independent of $P1_{1:0}$, $P2_{1:0}$. So the computation of q_{i+1} becomes

$$q_{i+1}=(S1[i]_1+S2[i]_1)+carry(S1[i]_0+S2[i]_0); \quad (4)$$

which can also be written as

$$q_{i+1}=(S1[i]_1 \oplus S2[i]_1) \oplus (S1[i]_0 \cdot S2[i]_0); \quad (5)$$

$A\_n_i=0$	$A\_n_i=1$
P1=0 P2=0	P1=B1_n P2=B2_n

2) $q_i=1$

$A\_n_i=0$	$A\_n_i=1$
P1=0 P2=n	P1=D1 P2=D2

When $A\_n_i=0$ then

$$q_{i+1} = ((S1[i]_1 + S2[i]_1 + P2_1) + carry (S1[i]_0 + S2[i]_0 + P2_0)) \bmod 2; \quad (6)$$

$$q_{i+1} = ((S1[i]_1 + S2[i]_1 + n_1) + carry (S1[i]_0 + S2[i]_0 + n_0)) \bmod 2; \quad (7)$$

The carry by adding zero bit of S1, S2 and n will be 1. Therefore above eq. can also be written as

$$q_{i+1}= (S1[i]_1 \oplus S2[i]_1 \oplus n_1 \oplus 1); \quad (8)$$

When $A\_n_i=1$ then the operands for addition to step 3b in algorithm 3 are D1 and D2. The two least significant bits (LSB) of D1 are 00. Also, the two LSB of D2 depend on two LSB of n as two LSB of B1_n and B2_n are 00.

So quotient q_{i+1} becomes

$$q_{i+1} = ((S1[i]_1 + S2[i]_1 + P1_1 + P2_1) + carry (S1[i]_0 + S2[i]_0 + P1_0 + P2_0)) \bmod 2; \quad (9)$$

As $P1_1=0$, $P1_0=0$, $P2_1= n_1$, $P2_0=n_0$

Therefore in this case eq (8) is used to calculate quotient.

The proposed Montgomery Modular Multiplication reduces the critical path of design by computing quotient for next iteration in parallel with carry save addition. The quotient computation requires a maximum delay of 2:1 MUX + 2 XOR + 1 AND delay. Due to parallel computation it does not add to path delay. Hence the critical path delay of our proposed design is 4:1 MUX + 2 Full adders.

IV. IMPLEMENTATION RESULTS AND COMPARISONS

The proposed Montgomery design and RSA Modular Exponentiation (Encryption) are coded in VHDL and synthesized in XILINX ISE 8.1i (Virtex 2) and XILINX ISE 12.4 (Virtex 5, device XC5VLX50 package FF1153 speed -3). The synthesis results of proposed Montgomery design for operand size 512, 1024 and 2048 bits are in Table 1 and Table 2. Area is in terms of number of slices in Table 1; slice registers, look up tables in Table 2.

Frequency is in MHz. Area and frequency (minimum period) are generated in synthesis report. Throughput is calculated as bit length multiplied by the frequency and divided by number of clock cycles. The proposed Montgomery design takes k+4 clock cycles where k is bit length of operands. RSA modular exponentiation (encryption) is implemented using proposed Modified Montgomery design. RSA modular exponentiation is done using both LSB (least significant bit) and MSB (most significant bit) binary method. Synthesis result for RSA MSB (H) and LSB (L) for 512 and 1024 bits are given in table 3 and 4. For implementation the bits of exponent e are taken 17 bits. RSA encryption can be done faster by choosing values for e=3, 17, or 65537 (2^{16}+1). Binary representation of 65537 is 10000000000000001. Taking this value and calculating the number of cycles of MMM in Modular exponentiation gives:

MSB design - 17 squarings +2 multiplications =19

Montgomery cycles in MSB = 19 × (K+4)

LSB design – 17 squarings +2 multiplications

(Squarings and multiplications can be done in parallel) = 17

Montgomery cycles in LSB=17 × (K+4)

Throughput in Table 3 and Table 4 is calculated using 19 × (K+4) MMM cycles for MSB and 17 × (K+4) MMM cycles for LSB.

The proposed Montgomery design has a higher frequency and throughput as compared to [7][2] and [8-11]. Throughput/Area (T/A) has been taken as a measure of efficiency. Our design has three times T/A as compared to design in [7][2]. Also T/A value for RSA exponentiation (LSB) using our design is almost double that of RSA implemented with [7][2].

Table 1: FPGA IMPLEMENTATION RESULTS OF MONTGOMERY MODULAR MULTIPLIERS (Virtex 2)

	Bit Length	FPGA Technology	Area (Slices)	Frequency (MHz)	Throughput Rate (Mbps)	Throughput/Area (Mbps/Slices)
[7][1]	512	XC2V1500	5170	126.71	126.46	.024
	1024	XC2V3000	10332	101.71	101.61	.009
	2048	XC2V6000	20986	90.09	90.05	.004
[7][2] (algo 2)	512	XC2V1500	5782	122.03	121.55	.021
	1024	XC2V3000	11520	111.32	111.1	.009
	2048	XC2V6000	23108	90.73	90.64	.003
[8]	512	XC2V1500	1678	89.3	29.71	.017
	1024	XC2V3000	3334	88.9	29.60	.008
	2048	XC2V6000	6782	87.1	29.02	.004
[9]	512	XC2V1500	3125	72.1	71.82	.022
	1024	XC2V3000	6243	79.2	79.05	.012
[11]	512	XC2V1500	3480	156.82	155.9	.044
	1024	XC2V3000	6953	136.45	136.05	.019
	2048	XC2V4000	14015	135.58	135.38	.009
Our	*512*	*XC2V1500*	*3547*	*235.81*	*233.98*	*.065*
	1024	*XC2V3000*	*6814*	*188.69*	*187.95*	*.027*
	2048	*XC2V4000*	*13753*	*185.33*	*184.96*	*.013*

Table 2: FPGA IMPLEMENTATION RESULTS OF MODIFIED MONTGOMERY MODULAR MULTIPLICATION (Virtex 5)

	Bits	FPGA Technology	Slice Registers	Slice LUTs	Freq (MHz)	Throughput (Mbps)
Our	512	XC5VLX50	4121	4652	430.84	427.5
	1024	XC5VLX50	8218	9252	422.24	420.59
	2048	XC5VLX50	16411	18468	418.05	417.23

Table 3: FPGA IMPLEMENTATION RESULTS OF RSA MODULAR EXPONENTIATION (Encryption) (Virtex 2)

	Bit Length	H / L	FPGA Technology	Area (Slices)	Frequency (MHz)	Throughput (Mbps)	Throughput /Area Kbps/slices
[7][1]	512	L	XC2V3000	11304	102.31	5.10	0.451
	1024	L	XC2V6000	23208	95.90	4.79	0.206
[11]	512	H	XC2V3000	6073	117.247	6.13	1.009
		L	XC2V3000	8962	117.564	6.87	0.766
	1024	H	XC2V3000	12040	109.636	5.75	0.477
		L	XC2V4000	17818	109.914	6.44	0.361
Our	512	H	XC2V3000	5777	150.655	7.86	1.36
		L	XC2V3000	8934	151.042	8.81	0.986
	1024	H	XC2V3000	11898	140.86	7.38	0.620
		L	XC2V4000	17770	141.205	8.27	0.465

Table 4: FPGA IMPLEMENTATION RESULTS FOR RSA MODULAR EXPONENTIATION (Encryption) (Virtex 5)

	Bit Length	H/L	FPGA Tech	Slice Registers	Slice LUTs	Freq (Mhz)	Throughput (Mbps)
Our	512	H	XC5VLX50	6773	6773	239.09	12.48
		L	XC5VLX50	9865	12433	239.29	13.96
	1024	H	XC5VLX50	13432	13437	229.49	12.03
		L	XC5VLX50	19597	25777	229.67	13.45

V. CONCLUSIONS

In this paper the critical path delay of Montgomery design is reduced by parallel and simplified quotient computation. This is done only at the expense of two extra bits and two extra iterations. Our design has a high frequency and high throughput. It can be used for both LSB and MSB methods for RSA exponentiation.

REFERENCES

[1] R. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Commun. ACM, vol. 21, issue 2, pp.120-126, Feb. 1978.

[2] P. L. Montgomery, Modular multiplication without trial division, Math. Comput., vol 44, pp. 519-521, Apr. 1985.

[3] C.D. Walter, Systolic modular multiplication, IEEE Trans. Comput., vol. 42, no. 3, pp 376-378, Mar. 1993.

[4] S.E. Eldridge and C.D. Walter, Hardware implementation of montgomery's modular multiplication algorithm, IEEE Trans. Comput., vol. 42, no. 6, pp. 693-699, Jun. 1993.

[5] C.D. Walter, Montgomery exponentiation needs no final subtractions, Electron. Lett., vol. 32, no. 21, pp. 1831-1832, Oct. 1999.

[6] C. McIvor, M. McLoone, and J.V. McCanny, Fast Montgomery modular multiplication and RSA cryptographic processor architectures, in Proc. 37th Asilomar Conf. Signals, Syst. Comput., Nov. 2003, vol. 1, pp. 379-384.

[7] C. McIvor, M. McLoone and J. V. McCanny, Modified Montgomery modular multiplication and RSA exponentiation techniques, in Proc. IEE Comput. Digit. Techniques, vol. 151,no. 6, pp. 402–408, Nov. 2004.

[8] K. Manochehri and S. Pourmozafari , Fast Montgomery modular multiplication by pipelined CSA architecture," in Proc.IEEE Int. Conf. Microelectron., Dec. 2004, pp.144-147.

[9] K. Manochehri and S. Pourmozafari, Modified Radix-2 Montgomery Modular Multiplication to Make It Faster and Simpler, Proc. Int. Conference on Information Technology: Coding and Computing, Apr. 2005, pp 598 – 602.

[10] M.D. Shieh, J.H. Chen, H.H Wu, W.C Lin, A New Modular Exponentiation Architecture for Efficient Design of RSA Cryptosystem, IEEE Trans. VLSI Sys. vol. 16, no. 9, pp 1151-1161, Sep. 2008.

[11] R. Verma, M. Dutta, R. Vig, Modified Montgomery Modular Multiplication for RSA Cryptosystem, International Journal of Computational Intelligence and Information Security, vol. 2, no. 9, pp 39-47, Sept. 2011.