World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:7, No:12, 2013

Modified Montgomery for RSA Cryptosystem

Rupali Verma, Maitreyee Dutta, Renu Vig

***Abstract*—**Encryption and decryption in RSA are done by modular exponentiation which is achieved by repeated modular multiplication. Hence efficiency of modular multiplication directly determines the efficiency of RSA cryptosystem. This paper designs a Modified Montgomery Modular Multiplication in which addition of operands is computed by 4:2 compressor. The basic logic operations in addition are partitioned over two iterations such that parallel computations are performed. This reduces the critical path delay of proposed Montgomery design. The proposed design and RSA are implemented on Virtex 2 and Virtex 5 FPGAs. The two factors partitioning and parallelism have improved the frequency and throughput of proposed design.

***Keywords*—**RSA, Montgomery modular multiplication, 4:2 compressor, FPGA.

I. Introduction

THE growth of data communications and other internet services has made security a vital service to electronic communications. Four basic services like confidentiality, integrity, authentication and non repudiation are required by communication systems. RSA is a popular public key cryptosystem for encryption and digital signatures [1]. Encryption, decryption and digital signatures are modular exponentiation operations which are achieved by repeated modular multiplications. The security of RSA cryptosystem depends on inability to efficiently factorize the modulus of size 1024 bits or more. Hence the security of RSA will be more if the operands are of large size. But for the large size operands, high throughput rate for RSA is hard to achieve. Many researchers have worked to improve the throughput of RSA by improving the modular multiplication design.

In 1985, P.L. Montgomery [2] proposed an efficient method for modular multiplication which replaces trial division by modulus with series of additions and shift operations. The critical operation in Montgomery design is addition of large size operands. To avoid the carry propagation during addition several architectures are proposed in literature such as systolic array modular multipliers [3], [4] and carry save adder (CSA) architectures [5]-[11]. McIvor et al. [5], [6] 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 and extra control logic). These designs take input (except the modulus) and give output in carry save format. This avoids repeated output/input format conversion, lengthy and costly conventional additions. Also, the critical path delay of Montgomery architectures with carry save adders is word length independent [6]. The New Montgomery multiplication proposed by K. Manochehri et al. [8] has higher throughput than [5], [6] as it calculates quotient in parallel with the summation of partial sum and partial product. H. Thapliyal et al. [9] have used 4:2 compressor and carry save adder in five-to-two CSA and novel hardware unit in 4:2 CSA to reduce the critical path delay. M.D. Shieh [11] proposed new modular exponentiation architecture with unified multiplication/square module in which the number of operands was reduced by mathematical manipulation. The critical path was reduced to 4 to 2 CSA with limited overhead but it could be used only for H algorithm of modular exponentiation.

The work in this paper is based on rearrangement of logic operations for addition with 4:2 compressor. It extends from existing 4:2 compressor applied in 4:2 CSA Montgomery [9]. Parallel and partitioned computations have reduced the critical path delay of proposed Montgomery modular multiplication. Our work has high throughput and is suitable for both H and L binary methods of RSA Modular Exponentiation. Section II gives brief introduction to Montgomery algorithms. Section III discusses the Modified Montgomery design. Section IV gives implementation results on FPGAs. Finally section V concludes the paper.

II. Montgomery Modular Multiplication

Montgomery modular multiplication treats the digits of multiplier from least significant bit to most significant bit, performs shift down of partial result instead of shift up, and add rather than subtracting multiples of the modulus. It is most efficient method for performing modular multiplication in which the quotient only depends on the least significant digit of operands and the time consuming division is avoided by shifting modular addition. It is suitable for both hardware and software implementations. This has led to extensive research in Montgomery modular multiplication and form the basis of most of the reported high performance RSA hardware architectures. Montgomery's algorithm computes modular multiplication with inputs A, B and n. Inputs A and B are in Montgomery's domain and n is k bit modulus. Then the output is S such that $S=A\times B\times r^{-1} \bmod n$ where $r=2^k$. The transformations between the n-residue (Montgomery domain) and the integer set are done using Montgomery's modular multiplication (MMM).

- A, the n residue of a is obtained as follows

$$MMM(a, r^2, n) = a\times r^2\times r^{-1} \bmod n$$

Rupali Verma is with PEC University of Technology, Chandigarh, India (e-mail: rupali@ pec.ac,in).
Maitreyee Dutta is with National Institute of Technical Teachers Training and Research, Chandigarh, India (e-mail: d_maitreyee@yahoo.co.in).
Renu Vig is with U.I.E.T, Panjab University, Chandigarh, India (e-mail: renuvig@hotmail.com).

$$A=a \times r \bmod n \quad (1)$$

- The n residue to integer value is obtained as follows

$$a=MMM(A, 1, n) = A\times 1\times r^{-1} \bmod n \quad (2)$$

Algorithm 1 MMM (A, B, n)

// Montgomery's modular multiplication algorithm
// Inputs: A and B (multiplier and multiplicand in
// Montgomery's domain, k bits)
// n (k bits modulus), A, B < n
// Output S= $A\times B\times r^{-1}$ mod n, where $r=2^k$

1. S=0;
2. for i=0 to k-1
3. { q[i]=(S+ $A_i \times B$) mod 2;
4. S= (S + $A_i \times B$ + q[i] × n)/2; }
5. if(S ≥ n)
6. S=S-n;
7. return S;

After k iterations if the final result S ≥ n then an extra operation S=S-n is needed. To remove this subtraction, C.D. Walter [12] kept the range of convergence at [0, 2n) and increased the number of iterations from k to k+2. Thus the bit width of the operands and the value of r were changed accordingly.

Algorithm 2 Walter MMM (A, B, n)

// Walter's Montgomery modular multiplication algorithm
// Inputs: A (multiplier in Montgomery's domain, k+2 bits
// $A_{k+1}=0$
// B (multiplicand in Montgomery's domain, k+1 bits)
// n (k bit modulus), A, B < 2n
// Output S= $A\times B\times r^{-1}$ mod n, where r= 2^{k+2},
// 0 ≤ S < 2n

1. S=0;
2. for i=0 to k+1
3. { q[i]= (S + $A_i \times B$) mod 2;
4. S= (S + $A_i \times B$ + q[i]×n)/2; }
5. return S;

To further simplify the quotient computation the multiplicand B can be shifted up by one bit to make $B_0=0$ [8]. The price for this simplification is one extra iteration by making $A_{k+2} = 0$.

Algorithm 3 Modified MMM (A, B, n)

// Montgomery's modular multiplication algorithm
// Inputs A (k+3 bit multiplier) $A_{k+2}=0$, $A_{k+1}= 0$
// B (k+2 bit multiplicand) $B_0=0$
// n (k bit modulus)
// Output S= $A\times B\times r^{-1}$ mod n, where r= 2^{k+3}

1. S=0;
2. for i=0 to k+2
3. { q[i]= S_0 mod 2;
4. S=(S + $A_i \times B$ + q[i]×n)/2; }
5. return S;

Step 4 in algorithm 3 requires addition of long operands. To avoid long carry propagation delay, carry save logic can be applied to solve the problem. Therefore step 3 and 4 become

$$q[i]= (S1_0 + S2_0) \bmod 2; \quad (3)$$

$$S1, S2 = (S1+S2+A_i \times (B1 + B2) + q[i]\times n)/2; \quad (4)$$

In [13] a Modified 4:2 CSA Montgomery has been presented where the input and output operands are in carry save format. Each iteration computes quotient for next iteration. It has two levels of carry save logic with additional control logic to determine the operands for addition. Algorithm 4 considers the Walter approach and extends the operands and bit width of r accordingly for the design in [13].

Algorithm 4 Modified 4:2 Carry Save Montgomery Modular Multiplication (A1, A2, B1, B2, n)

//Inputs A1, A2 (carry save format of multiplier, k+3 bits.
// $A1_{k+2}=0$, $A2_{k+2}=0$, $A1_{k+1}=0$, $A2_{k+1}=0$
// B1, B2 (k+2 bit multiplicand $B1_0=0$, $B2_0=0$
// n (k bits modulus), $r=2^{k+3}$
// Output S1[k+3], S2[k+3]
// = (A1+A2) × (B1+B2) × r^{-1} mod n
1a. (S1[0],S2[0])=(0,0)
1b. P1[-1]=0, P2[-1]=0;
1c. carry=0;
1d. D1, D2= CSR (B1+B2+n)
/*Pre-computation of operands */
1e. q[0]=0; // $S1[0]_0 \oplus S2[0]_0$
1f. (carry, $A_0 = A1_0 + A2_0$ + carry);
2a. for i in 0 to k+2 loop
2b. if (A_i=0 and q[i] = 0) then P1[i]=0, P2[i]=0;
elsif(A_i=1 and q[i]=0) then P1[i]=B1, P2[i]=B2;
elsif(A_i=0 and q[i]=1) then P1[i]=0, P2[i]=n;
else P1[i]=D1, P2[i]=D2;
endif;
2c. S1[i+1], S2[i+1]
=CSR(S1[i]+S2[i]+P1[i]+P2[i])/2;
2d. q[i+1]=($S1[i+1]_0 \oplus S2[i+1]_0$) ;
2e. (carry, A_{i+1})=$A1_{i+1}$+$A2_{i+1}$+carry);
end loop;
/* Steps 2a-2d are parallel to step 2e */
3.return (S1[k+3], S2[k+3]);

The critical path delay in algorithm 4 is 4:1 MUX + 2 Full Adders+1XOR. It requires only 1 XOR delay to compute quotient whereas 4:2 and 5:2 CSA Montgomery designs [5], [6] have quotient delay of 2 XOR + 1 AND. In [9], the authors used 4:2 compressor to reduce critical path delay in Montgomery designs [5], [6]. The addition operation S1[i] + S2[i] + P1[i] + P2[i] with 4:2 compressor [14] is shown in Fig 1. In this paper, a Modified Montgomery design is proposed which uses 4:2 compressor for addition of operands S1[i]+S2[i]+P1[i]+P2[i] such that logic operations are partitioned over two iterations to perform parallel computations.

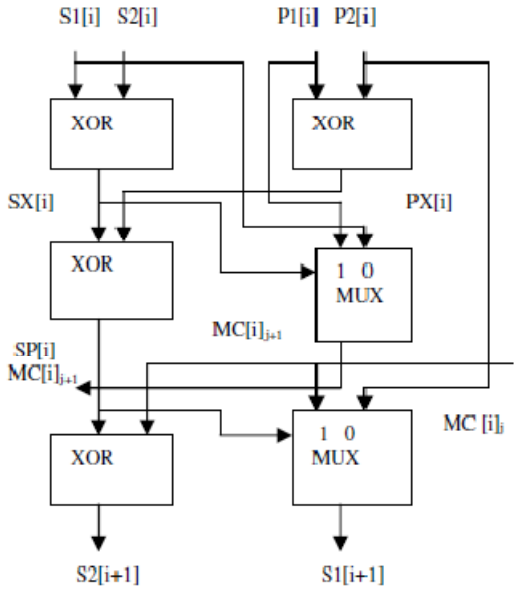

Fig. 1 Addition with 4:2 compressor

III. Proposed Montgomery Modular Multiplication

The features of proposed design are:

(1) There are 2 phases: pre-computation and computation.

Modular Multiplication iterations span from i =0 to k+2 with additional iteration i= -1. During this iteration shift down of result gives the same value 0

(2) In pre-computation three operands B1, B2 and n are added. Intermediate result BX is stored in register. Also DX is computed. BX and DX are used when operands are added.

(3) Each i^{th} iteration computes S1[i+1], S2[i+1]= S1[i]+S2[i]+P1[i]+P2[i] with 4:2 compressor by applying 2P approach: parallelism and partitioning.

(4) Each iteration computes quotient for next iteration q[i+1] and multiplier bit A_{i+2}.

(5) Also each iteration computes intermediate results SP[i], MC[i] and SX[i+1] and PX[i+1]

(6) Multiplier bit A_{i+1} and quotient bit q[i+1] determines operands P1[i+1], P2[i+1] and PX[i+1] for next iteration.

Algorithm 5 Proposed Modified 4:2 Compressor Montgomery Modular Multiplication (A1, A2, B1, B2, n)

// Inputs: A1, A2 (carry save format of multiplier,
// k+3 bits)
// $A1_{k+2}$=0, $A2_{k+2}$=0, $A1_{k+1}$=0, $A2_{k+1}$=0
// B1, B2 (k+2 bit multiplicand) $B1_0$= 0, $B2_0$= 0
// n (k bits modulus), $r=2^{k+3}$
// Output
//S1[k+3],S2[k+3]=(A1+A2)×(B1+B2)×r^{-1} mod n
1.1a. (S1[-1], S2[-1])= (0,0)
1b. P1[-1]=0, P2[-1]=0, PX[-1]=0, SX[-1]=0
1c. carry=0
2. 2a. BX=B1⊕ B2;
2b. D1= **(BX • n) + ($\overline{BX}$ • B1);**
2c. D2= BX ⊕ n;
2d. DX= D1 ⊕ D2;
/* D1, D2=CSR (B1 + B2 + n) */
2e. carry, A_0 =$A1_0$ + $A2_0$ + carry;
3. for i in -1 to k+2 loop
3a. SP[i] = SX[i] ⊕ PX[i];
parallel
MC[i] = (SX[i] • P1[i]) + ($\overline{SX[i]}$ •S1[i]) ;
3b. S2[i+1]= (SP[i] ⊕ (MC[i],0))/2 ;
parallel
S1[i+1]=((SP[i]• (MC[i],0)) + ($\overline{SP[i]}$ •P2[i]))/2;
3c. SX[i+1]= **S1[i+1] ⊕ S2[i+1];**
parallel
{ q[i+1] =$S1[i+1]_0$ ⊕ $S2[i+1]_0$;
if (A_{i+1}=0 and q[i+1] =0)
then P1[i+1]=0; P2[i+1]=0; PX[i+1]=0;
elsif (A_{i+1}=1 and q[i+1] =0)
then P1[i+1]=B1; P2[i+1]=B2; PX[i+1]=BX;
elsif(A_{i+1}=0 and q[i+1] =1)
then P1[i+1]=0; P2[i+1]= n; PX[i+1]= n;
else P1[i+1]=D1; P2[i+1]=D2; PX[i+1]=DX
end if;
}
parallel
carry, A_{i+2} =$A1_{i+2}$ + $A2_{i+2}$ + carry;
4. Return (S1 [k+3], S2[k+3]);
/* Steps 3a to 3c are parallel with computation of A_{i+2}*/

During pre-computation, D1 and D2 are computed with multiplexer in carry path. Intermediate result, BX is used for 4:2 addition when multiplier bit is 1 and quotient bit is 0. Also DX is computed and used when both multiplier and quotient bits are 1.

Step 3 of algorithm 5 iterates to perform the computations. Each iteration performs lower 2 levels of addition and first level of addition for next iteration of Fig. 1. Based on quotient and multiplier bits of next iteration P1, P2 and PX are determined.

- P1[i+1]=0 , P2[i+1]= 0
  then PX[i+1]=0 (0 ⊕ 0=0)
- P1[i+1]=B1 , P2[i+1]=B2
  then PX[i+1]=BX (B1 ⊕ B2=BX)
- P1[i+1]=0 , P2[i+1]=n
  then PX[i+1]=n (0 ⊕ n=n)
- P1[i+1]=D1, P2[i+1]=D2
  then PX[i+1]=DX (D1 ⊕ D2=DX)

IV. Implementation Results and Analysis

A. Synthesis Results

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 XC5VLX50 package FF1153 speed -3). The synthesis results of proposed Montgomery design for operand size 512, 1024 and 2048 bits are in Tables I-III. Area is in terms of number of slices. 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+6 clock cycles where k is bit length of operands. Table I give detailed area results on Virtex 2 and Virtex 5 FPGAs.

TABLE I
AREA RESULTS FOR PROPOSED MONTGOMERY ON VIRTEX 2 FPGAS

Bit Len	FPGA Technology	Slices	Slice Flip Flops	LUTs	Freq
512	XC2V1500	4211	5288	7955	253.7
1024	XC2V3000	9213	10963	16917	250.5
2048	XC2V4000	17140	21544	32001	246.3

TABLE II
AREA RESULTS FOR PROPOSED MONTGOMERY ON VIRTEX 5 FPGAS

Bit Len	FPGA Tech	Slice Flip Flops	Slice LUTs	Freq	Throughput
512	XC5VLX50	5142	4637	534.7	528.5
1024	XC5VLX50	10267	9247	530.7	527.6
2048	XC5VLX50	20509	18463	530.7	529.1

Tables I and II show that when the proposed design is implemented on Virtex 2 and Virtex 5 FPGAs, there is little difference in number of slice flip flops but the difference in LUTs is large. Each slice on Virtex 2 FPGA has two 4 input function generator which can be programmed as 4 input LUTs. Whereas there are four 6 input LUTs in SLICEL and SLICEM on Virtex 5 FPGAs. Therefore to implement a function on Virtex 2 requires more LUT combining and hence more number of LUTs to implement a design. The frequency results are more than double on Virtex 5 FPGAs. This is due to improved version of wiring architecture named "Diagonally symmetric interconnect pattern" [15]. This type of architecture reduces the delays, thus making more logic blocks accessible with a smaller number of hops (switch blocks).

TABLE III
FPGA IMPLEMENTATION OF MONTGOMERY MODULAR MULTIPLIERS

	Bit Len	FPGA Tech	Area ,A (Slices)	Freq (MHz)	Throughput Rate, T (Mbps)	T/A (Mbps/ Slices)
[6][1]	512	XC2V1500	5170	126.7	126.4	.024
	1024	XC2V3000	10332	101.7	101.6	.009
[6][2]	512	XC2V1500	5782	122.03	121.5	.021
	1024	XC2V3000	11520	111.3	111.1	.009
[7]	512	XC2V1500	1678	89.3	29.71	.017
	1024	XC2V3000	3334	88.9	29.60	.008
	2048	XC2V6000	6782	87.1	29.02	.004
[8]	512	XC2V1500	3125	72.1	71.82	.022
	1024	XC2V3000	6243	79.2	79.05	.012
[10]	512	XC2V3000	2902	121.5	120.3	.041
	1024	XC2V3000	4512	114.2	113.4	.025
[11]	512	XC2V1500	4029	221.9	220.2	.054
	1024	XC2V3000	8000	219.06	218.2	.027
our	512	XC2V1500	4211	253.7	250.8	.059
	1024	XC2V3000	9213	250.5	249.07	.027
	2048	XC2V4000	17140	246.3	245.61	.014

[1]: designed with Five-to-two CSA
[2]: designed with four-to-two CSA

Table III compares the results with related implementation on Virtex 2 FPGA. It shows that proposed design has more than double the frequency and throughput when compared to 4:2 CSA Montgomery [6][2] proposed by McIvor. This is because the critical path delay of proposed design is 3 XOR+4:1 MUX as compared to 4:1MUX+2 Full adders + 2 XOR+ 1 AND of [6][2]. Proposed Montgomery design has highest frequency amongst the designs where no output/input format conversion is done at the end of modular multiplication [5]-[9], [11]. When compared with [10] where the format conversion is done using carry save addition, our frequency and throughput is double at the cost of area. The design in [16] is a semi-systolic. Also the approach in [11], [16] are only applicable for RSA MSB method. The proposed design can be used in both RSA MSB and LSB binary modular exponentiation methods. Table III shows our design has highest throughput/area ration for 512 bits.

TABLE IV
RSA MODULAR EXPONENTIATION ON VIRTEX 2 FPGAS

Bit Len	H/L	FPGA Technology	Slices	Slice Flip Flops	LUTs	Freq
512	H	XC2V3000	6270	7960	11760	224.6
	L	XC2V3000	11290	13506	21488	224.5
1024	H	XC2V6000	12930	16296	24005	222.6
	L	XC2V6000	24259	28231	44829	222.1

TABLE V
RSA MODULAR EXPONENTIATION ON VIRTEX 5 FPGAS

Bit Len	H/L	FPGA Technology	Slice Flip Flops	LUTs	Freq
512	H	XC5VLX50	7795	7479	491.64
	L	XC5VLX50	12932	13124	496.03
1024	H	XC5VLX50	15478	14888	441.2
	L	XC5VLX50	25736	27137	445.05

TABLE VI
RESULTS OF RSA MODULAR EXPONENTIATION

	Bit Len	H/L	FPGA	Area, A (Slices)	Fre (MHz)	Throughput, T (Mbps)	T/A Kbps/slices
[6][1]	512	L	XC2V3000	11304	102.31	5.10	0.45
	1024	L	XC2V6000	23208	95.90	4.79	0.20
[11]	512	H	XC2V3000	6294	168.38	9.28	1.47
	1024	H	XC2V6000	12537	152.49	8.44	0.67
[16]	512	H	XC2V3000	7832	218.72	11.87	1.51
	1024	H	XC2V6000	15826	215.83	11.85	0.74
Our	512	H	XC2V3000	6270	224.65	11.68	1.86
		L	XC2V3000	11290	224.58	13.05	1.15
	1024	H	XC2V6000	12930	222.69	11.65	0.90
		L	XC2V6000	24259	222.12	12.98	0.53

[1]: designed with Five-to-two CSA

RSA modular exponentiation (encryption) is implemented using proposed Modified Montgomery design. Both LSB and MSB binary method of RSA are coded in VHDL. Synthesis result for RSA MSB (H) and LSB (L) for 512 and 1024 bits are given in Tables IV-VI. Detailed results for RSA modular exponentiation are given in Tables IV and V. For implementation the bits for exponent are taken 17 bits. Tables IV and V show the difference in LUTs and frequency due to different FPGA technology. The RSA design in this paper has frequency higher than [6], [11] and [16]. Also area results for RSA MSB (H) are lower than [16] since we are reusing registers for storing the final result instead of new registers (only in MSB method). RSA encryption can be done faster by choosing the values for e=3, 17, or 65537(2^{16}+1) [17]. Binary representation of 65537 is 10000000000000001. Taking this value and calculating the number of Montgomery modular multiplication (MMM) cycles in modular exponentiation gives:

MSB design- 17 squarings + 2 multiplications= 19

Montgomery cycles in MSB=19 × (k + 6)

LSB design- 17 squarings + 2 multiplications

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

Montgomery cycles in LSB=17 × (k+6)

Throughput of RSA given in tables is calculated using 19 × (k+6) MMM cycles for MSB and 17 × (k+6) MMM cycles for LSB binary modular exponentiation. The throughput in [16] is better than our RSA design. The H algorithm of binary method for RSA modular exponentiation in [11], [16] adds carry save values of message M to get binary M. This requires additional conversion cycles. Thus our overall performance will be better when compared with [16]. If throughput / area is taken as efficiency factor then our RSA design has highest value among related designs as shown in Table VI.

V. CONCLUSIONS

The proposed Montgomery modular multiplication partitions the logic operations of addition with 4:2 compressor which facilitates parallel computations and reduced delay. Though the area requirements of design are high but the proposed design has high throughput /area ratio. It is suitable for applications where high throughput is required and area is not a limitation.

REFERENCES

[1] R. Rivest et al., "A method for obtaining digital signatures and public key cryptosystems," Commun. ACM, vol 21, issue 2, Feb 1978, pp. 120-126.
[2] P.L. Montgomery, "Modular multiplication without trial division," Math Comput, vol 44, Apr. 1985, pp. 519-521.
[3] C.D. Walter, "Systolic modular multiplication," IEEE Trans. Comput, vol 42, no 3, Mar 1993, pp. 376-378.
[4] S.E. Elridge et al., "Hardware implementation of Montgomery's modular multiplication algorithm," IEEE Trans. Comput., vol. 42, no. 6, Jun 1993, pp. 693-699.
[5] C. McIvor et al., "Fast Montgomery modular multiplication and RSA Cryptographic processor architectures," Proc. 37th Asilomar Conf. Signals, Syst. Comput., vol. 1, Nov. 2003, pp. 379-384.
[6] C. McIvor et al., "Modified Montgomery modular multiplication and RSA exponentiation techniques," Proc. IEEE Comput. Digit. Techniques, vol. 151, no.6, Nov. 2004, pp. 402-408.
[7] K. Manochehri et al., "Fast Montgomery modular multiplication by pipelined CSA architecture," Proc. IEEE Int. Conf. Microelectron, Dec. 2004, pp. 144-147.
[8] K. Manochehri et al., "Modified Radix 2 Montgomery Modular Multiplication to Make It Faster and Simpler," In Proc. Int. Conference on Information Technology: Coding and Computing, Apr 2005, pp. 598-602.
[9] H. Thapliyal et al., "Modified Montgomery Modular Multiplication Using 4:2 Compressor and CSA Adder," In Proc. Of Third Int. Workshop on Electronic Design, Test and Applications, 2005.
[10] Y. Y Zhang et al., "An efficient CSA architecture for Montgomery modular multiplication," Microprocessors and Microsystems, vol 31, no. 7, Nov.2007, pp. 456-459.
[11] M.D. Shieh et al., "A New Modular Exponentiation Architecture for Efficient Design of RSA Cryptosystem," IEEE Trans. On Very Large Scale Integration Systems, vol. 16, no. 9, Sept 2008, pp. 1151-1161.
[12] C.D. Walter, "Montgomery exponentiation needs no final subtractions," Electron. Lett., vol. 32, no. 21, Oct. 1999, pp. 1831-1832.
[13] R. Verma et al., "Modified Montgomery Modular Multiplication for RSA Cryptosystem," Int. Journal of Computational Intelligence and Information Security, vol 2, no. 9., Sept 2011, pp. 39-47.
[14] S. Veeramachaneni et al., "Novel Architectures for High Speed and Low Power 3-2, 4-2 and 5-2 Compressors," In 20th Int. Conf. on VLSI design, 2007.
[15] P.B. Minev et al., "The Virtex 5 Routing and Logic Architecture," Electronics-ET 2009, 14-17 Sept, Sozopol, Bulgaria.
[16] M.D. Shieh et al., "A New Algorithm for High Speed Modular Multiplication Design," IEEE Trans. On Circuits and Systems-I: Regular Papers, vol. 56, no. 9, Sept. 2009, pp. 2009-2019.
[17] B. Schneier, *Applied Cryptography Protocols, Algorithms and Source Code in C*: Second edition, Wiley.