VLSI Architecture for Montgomery Modular Multiplication using Ripple Carry Adder

M.Shunmugalakshmi[1] M. Bhuvaneswari[2] E. Dharani[3] R. Dhivyaa[4] I. Jasmine silviya[5]
[1]Assistant Professor [2,3,4,5]UG Scholars
[1,2,3,4,5]SNS College of Engineering, Coimbatore, India.

Abstract— This paper proposes a RCA(ripple carry adder) based Montgomery multiplier to achieve low cost and high performance. The proposed multiplier uses ripple carry adder to avoid the high usage of memory in the system. This RCA is used to do the addition operation at each stage without saving the carry but by propagating it to the next stage and also to perform operand precomputation and format conversion that helps to convert carry save format to binary format leading to a low hardware complexity and high speed performance at the expense of more clock cycles for completing single multiplication. To overcome the weakness, a RCA is proposed to reduce the additional clock cycles for operand precomputation. Experimental results show that the proposed Montgomery modular can achieve higher performance and significant hardware complexity reduction when compared with previous design.

Key words: Ripple carry addition, low cost architecture, Montgomery modular multiplier, cryptosystem

I. INTRODUCTION

Now-a-days, the arithmetic operation is the base for many kinds of innovative operations. Addition, subtraction, multiplication and division are the basic operations in arithmetic. Among these multiplication and addition are the frequently used. Multiplication is done to add a integer number to a particular number of times by itself. Multiplication can be done with two numbers which are called as multiplier and multiplicand. The Multiplication is done by adding a number that is multiplicand to itself for a number of times, by another number that is Multiplier. The product value is generated as output. Each digit of the multiplier is multiplied with the multiplicand beginning with the LSD. Intermediate results are produced and are called as partial products. The sum of all the partial-products gives the final product value. The steps involved in multiplication are partial product generation, partial product reduction and final addition. When N-bit multiplicand multiplied with with an M-bit multiplier, M partial products are generated and N+ M bits long product is formed. Digital multipliers are used I digital design as they are fast, reliable and more efficient. They can do multiple operations with reduced number of components. Various multipliers are available depending upon the arrangement of the components. Multiplier architecture is chosen based on the application required. In most of the public key cryptosystem, modular multiplication becomes more critical with large integers. Therefore many algorithms and hardware implementation have been proposed to carry out the MM more quickly and Montgomery algorithm is one of the well-known MM algorithms. It determines the quotient depending on the last significant digit of operands and the complicated division is replaced in conventional MM with a number of shifting modular additions to generate S = A*B*R^-1(Mod N), where N is the K bit modules, R^-1 is the inverse of R module N, and R= 2^k mod N. As a result, this algorithm can be easily implemented in VLSI to enhance the speed of the encryption/decryption process. The three operand addition in the Montgomery algorithm iteration loop requires long carry propagation incase of large operands in binary representation. To solve this problem, many approaches based on carry save addition was proposed earlier. But using carry save addition leads to more delay and hardware complexity. This paper proposes a ripple carry addition method to carry out Montgomery multiplication to overcome the weakness appeared in previous design. The remainder of this paper is organized as follows. Section II briefly reviews CSA based Montgomery Multiplier. In Section III, we propose a simple and efficient RCA based Montgomery multiplier and its hardware architecture and in Section IV, experimental results were shown. Finally, the conclusion is drawn in Section V.

II. MONTGOMERY ALGORITHM BASED ON CSA ARCHITECTURE

The modular product S of A and B can be obtained as S=A*B*R^-1 (mod N), where R^-1 is the inverse of R modulo N. That is, R*R^-1=1 (mod N). In addition, the notation Xi-j indicates a segment of X from the ith bit to jth bit. Since the convergence range of Sin MM algorithm is 0<S<2N,an additional operation S=S-N is required to remove the oversize residue if S>N. Nevertheless, the long carry propagation for the very large operand addition still restricts the performance of MM algorithm.

A. SCS based Montgomery Multiplication:

In CSA based MM the intermediate result S of shifting modular addition can be kept in the carry-save representation (SS,SC). Note that the number of iterations in has been changed from k to k+2 to remove the final comparison and subtraction. However the format conversion from the carry-save format of the final modular product into its binary format is needed. The architecture of SCS-based MM is composed of one-level CSA architecture and one format converter. A 32-bit CPA with multiplexers and registers (denoted as CPA_FC), which adds two 32-bit inputs and generates a 32-bit output at every clock cycle, was adopted for the format conversion. Therefore, the 32-bit CPA_FC will take 32 clock cycles to complete the format conversion of a 1024-bit SCS-based Montgomery multiplication. The extra CPA_FC probably enlarges the area and the critical path of the CSA-MM- multiplier.

The precomputed D=B+N so that the computation of Ai*B+qi*N can be simplified into one selection operation. One of the operands 0,N,B, and D will be chosen if(Ai,qi)=(0,0),(0,1),(1,0), and (1,1), respectively. That is, S[k+2]=SS[k+2]+SC[k+2] is replaced with the repeated aarry-save addition operation (SS[k+2],SC[k+2])=SS[k+2]+SC[k+2] until SC[k+2]=0.

Note that the select signals of multiplier M1 and M2 generated by the control part are not for the sake of simplicity.

However, the extra clock cycles for format conversion are dependent on the longest carry propagation chain in SS[k+2]+SC[k+2]and about k/2 clock cycles are required in the CSA architecture.

Algorithm: Fig. 1: CSA based Montgomery Multiplier Algorithm.

Inputs: A, B, N (modules)
Output: SS [k=2]

1) (SS, SC) = (B+N+0);
2) while (SC ! = 0)
3) (SS, SC) = (SS+SC+0);
4) D = SS;
5) SS [0] = 0; SC [0] = 0;
6) fori =0 to k+1{
7) qi = (SS[i]0 + SC[i]0 + Ai*B0) mod 2;
8) if (Ai = 0 and qi = 0) x = 0;
9) if (Ai = 0 and qi = 1) x = N;
10) if (Ai = 1 and qi = 0) x = B;
11) if (Ai = 1 and qi = 1) x = D;
12) (SS[i+1], SC[i+1]) = (SS[i] + SC[i] + x)/2;
13) 13.}
14) while (SC[k+2] ! = 0)
15) (SS[k+2], SC[k+2]) = (SS[k+2] + SC[k+2] + 0);
16) return SS[k+2]

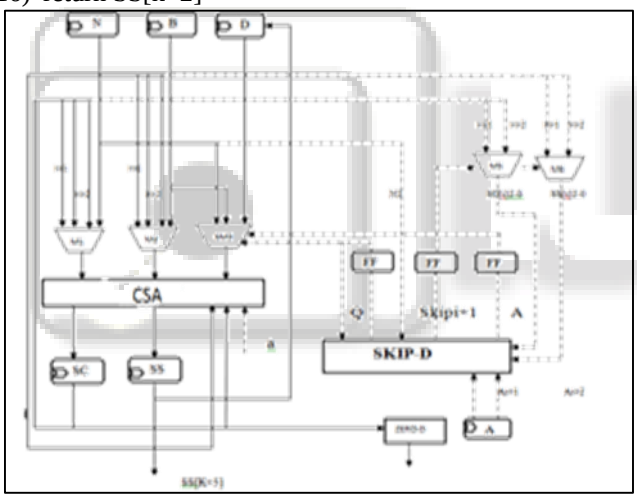

Fig. 2: CSA based Montgomery Multiplier architecture.

III. PROPOSED MONTGOMERY MULTIPLIER

In this section, we propose a new RCA based Montgomery MM to reduce the critical path delay of Montgomery multiplier. In addition, the drawback of additional clock cycles for completing one multiplication is also improved along with the advantages of small delay and low hardware complexity. The delay of CSA based multiplier can be reduced by using the RCA architecture. That is we can precompute D = B+N and the format conversion. To decrease the clock cycle number, a RCA architecture which can perform one three input carry save addition or two serial two input carry save additions is proposed to substitute for CSA architecture. Note that M3 has been replaced by SM3 in the proposed RCA architecture. In addition, we also skip the unnecessary operations in the for loop to further decrease the clock cycle for completing one Montgomery MM.

On the basis of delay reduction, clock cycle number reduction and quotient precomputation mentioned above, a new Montgomery MM algorithm using RCA architecture is proposed to significantly reduce the required clock cycles for completing one MM. As shown in MM-New algorithm, steps 1-5 for producing B^ and D^ are first performed. Note that because qi+1 and qi+2 must be generated in the ith iteration, the iterative index i of Montgomery MM will start from -1 instead of 0 and the corresponding initial values of q and A must be set to 0. Furthermore, the original for loop is replaced with the while loop in MM-New algorithm to skip some unnecessary iterations when skipi+1 = 1. In addition, the ending number of iteration in MM-New algorithm is changed to k+4 instead of k+1. This is because B is replaced with B^ and thus three extra iterations for computing division by two necessary to ensure the correctness of Montgomery MM. in the while loop, steps 8-12 will be performed in the proposed RCA architecture with one 4-to-1 multiplexer. The computations of qi+1, qi+2, and skipi+1 in step 13 and the selections of A^, q^, and i in steps 14-20 can be carried out in parallel with steps 8-12. Note that the right-shift operations of steps 12 and 15 will be delayed to next clock cycle to reduce the critical path delay of corresponding hardware architecture.

The hardware architecture of MM-New algorithm, denoted as MM-New multiplier, which consists of one RCA architecture, two 4-to-1 multiplexers, one simplified Multiplier, one skip detector skip, one zero detector and six registers. Skip_D is developed to generate skip i+1,q and A in the ith iteration. Both M4 and M5 are 3-bity 2-to-1 multiplexers and they are much smaller than k-bit multiplexers M1,M2 and SM3.In addition, the area of skip_D is negligible when compared with that of the k-bit RCA architecture. Similar to select signals of multiplexers M1 and M2 are generated by the control part, which are not depicted for the sake of simplicity.

At the beginning of Montgomery multiplication, the FFs stored skip i+1, q, A are first reset to 0 in step 1 of MM-new algorithm so that D=B+N can be computed via the RCA architecture. When performing the while loop, the skip detector skip-D is used to produce skip i+1, q and A. The skip_D is composed of four XOR gates, three AND gates, one NOR gate, and two 2-to-1 multiplexers. It first generates the q i+1,q i+2 and skip i+1 signal in the ith iteration and the selects the correct q and A according to skip i+1.At the end of ith iteration, q, A , and skip i+1 must be stored to FFs. In the next clock cycle of the ith iterations, SM3 outputs a proper x according to q and A generated in the ith iteration as shown in steps 8-to-11, and M1 and M2 output the correct SC and SS according to skip i+1 generated in the ith iteration.If skip i+1=0,SC>>1 and SS >>1 are selected. Otherwise, SC>>2 and SS>>2 are selected. That is, the right-shift 1-bit operations in steps 12 and 15 of MM new algorithm are performed together in the next clock cycle of ith iteration i. In addition, M4 and M5 also selected and output the correct SC[i]2:0 and SS[i]2:0 according to skip i+1 generated in the ith iteration. Note that SC[i]2:0 and SS[i]2:0 can also be obtained from M1 and M2 but a longer delay is required because they are 4-to-1 multiplexers. After the while loop in step 7-to-21 is completed, q and A stored in FFs are reset to 0.Then, the format conversions in steps 23 and 24 can be performed by the MM new multiplier similar to the computation of D=B+N in steps 3 and 4. Finally, SS[k+5] in binary format is outputted when SC[k+5]=0.

Algorithm: Fig. 3: Proposed Montgomery Multiplier algorithm.

Inputs: A,B,N (new modulus)
Output: SS[k+5]

1) B=B<<3;q=0; A=0; skipi+1=0;
2) (SS,SC)= 1F_CSA(B,N,O);
3) while (SC!=0)
4) (SS,SC)= 2H_CSA(SS,SC);
5) D=SS;
6) i=-1; SS[-1]=0; SC[-1]=0;
7) while(i<=k+4) {
8) if (A=0 and q=0) x=0;
9) if (A=0 and q=1) x=N;
10) if (A=1 and q=0) x=B;
11) if (A=1 and q=1) x=D;
12) (SS[i+1], SC[i+1]) = 1F_CSA(SS[i], SC[i], x)>>1;
13) compute qi+1; qi+2; and skipi+1 by (5),(7) and (8);
14) if (skipi+1 =1) {
15) SS[i+2] = SS[i+1]>>1; SC[i+2] = SC[i+1]>>1;
16) q= qi+2; A=Ai+2; i=i+2;
17) }
18) else {
19) q=qi+1; A=Ai+1; i=i+1;
20) }
21) }
22) q=0; A=0;
23) while (SC[k+5] !=0)
24) (SS[k+5], SC[k+5]) = 2H_CSA(SS[k+5], SC[k+5]);
return SS[k+5];

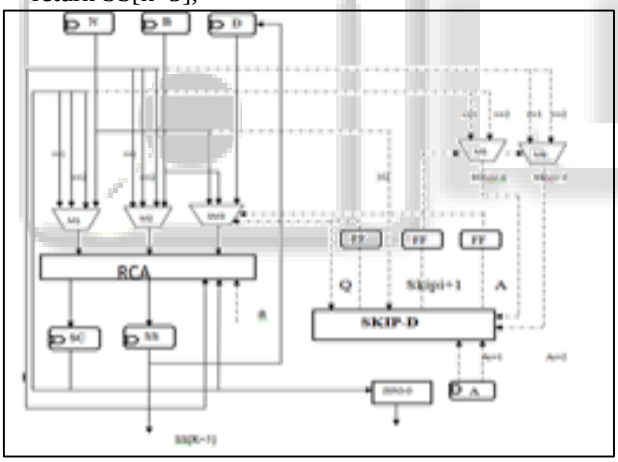

Fig. 4: Proposed Montgomery Multiplier architecture.

IV. Experimental Results

The parameters such as delay, circuit complexity, and power consumption are taken into consideration for proposed RCA based MM.

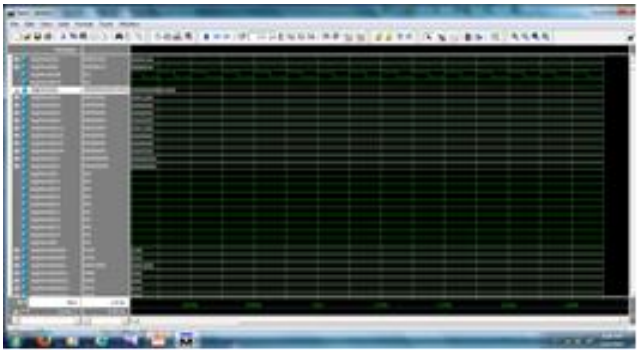

Fig. 5: Simulated output

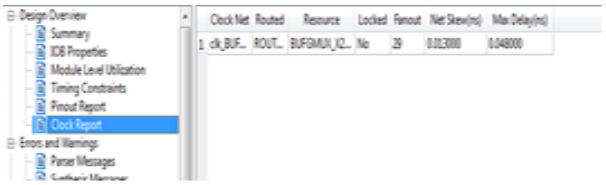

Fig. 6: Maximum delay calculation.

Fig. 7: Power calculation

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	28	1,920	1%
Number of 4 input LUTs	155	1,920	8%
Number of occupied Slices	88	960	9%
Number of Slices containing only related logic	88	88	100%
Number of Slices containing unrelated logic	0	88	0%
Total Number of 4 input LUTs	155	1,920	8%

Fig. 8: Device utilization.

V. Conclusion

FCS-based multipliers maintain the input and output operands of the Montgomery MM in the carry-save format to escape from conversion, leading to fewer clock cycles but larger area than SCS-based multiplier. To enhance the performance of Montgomery MM while maintaining the low hardware complexity, this paper has modified the SCS-based Montgomery multiplication algorithm and proposed a low-cost and high-performance Montgomery modular multiplier. The proposed multiplier used RCA architecture and skipped the unnecessary carry-save addition operations to largely reduce the delay and required clock cycles for completing one MM operation. Experimental results showed that the proposed approaches are indeed capable of enhancing the performance of radix-2 RCA based Montgomery multiplier while maintaining low hardware complexity.

References

[1] R.L.Rivest ,A.Shamir , and L.Adleman , “A method for obtaining digital signatures and public-key cryptosystem ,”Commun. ACM , vol. 21,no.2,pp.120-126,feb.1978.
[2] V.S.Miller ,”Use of elliptic curves in cryptography ,”in Advances in cryptology.Berlin,Germany: Springer-Verlag,1986,pp.417-426..
[3] N. Kblitz, “Elliptic curve cryptosystems,” Math. Comput., vol. 48, no177, pp. 203-209, 1987.
[4] P. L. Montgomery, “Modular multiplication without trial division,” Math. Comput., vol. 44, no. 170, pp. 519-521, Apr. 1985.
[5] Y. S. Kim, W. S. Kang, and J. R. Choi, “Asynchronous implementation of 1024-bit modular processor for RSA

cryptosystem," in Proc. 2nd IEEE Asia-Pacific conf. ASIC, Aug. 2000, pp. 187-190.
[6] V. Bunimov, M. Schimmler, and B. Tolg, "A Complexity-effective version of Montgomery's algorithm," in proc. Workshop Complex. Effective Designs, May 2002.
[7] Y.-Y. Zhang , Z. Li, L. Yang, and S. W. Zhang, "An efficient CSA architecture for Montgomery modular multiplication," Microprocessor microsyst., vol. 31, no. 7, pp. 456-459, Nov. 2007.
[8] C. D. Walter, "Montgomery exponentiation needs no final subtraction," Electron. Lett., vol. 35, no. 21, pp. 1831-1832, Oct.1999.
[9] P. Amberg, N. Pinckney, and D. M. Harris, "Parellel high radix Montgomery multipliers," in Proc. 42nd Asilomar Conf. Signals, Syst., Comput., Oct. 2008, pp. 772-776.
[10] F. Gang, "Design of modular multiplier based on improved Montgomery aligorithm and systolic array," in Proc. 1st Int. Multi-Symp. Comput. Comput. Sci., Jun. 2006, pp. 356-359.

