International Journal of Engineering Research
Volume No.5 Issue: Special 5, pp: 992-1128

ISSN: 2319-6890) (online), 2347-5013(print)
20 May 2016

Efficiency and performance review of Montgomery modular multiplication based on VLSI architecture

Ebin Geo Johnson
Electronics and Communication
Reva Institute of Technology, Bangalore, India 560064
reachebin@gmail.com

Neethu K
Asst. Prof
Department of Electronics and Communication
Reva UniversityBangalore, India 560064

Abstract— **Various Modular multiplication techniques are developed for the multiplication of large integers. In this paper the focus is mainly on the Montgomery algorithm for modular multiplication (MM), the basic Montgomery's algorithm that has been modified for improvement in performance and area time product in different designs are reviewed, and the improvements made in the algorithm of each designs are traced in a sequential manner from the initial. This starts with the various alterations in the basic algorithm using carry save adders (CSA) to eliminate the carry propagation that are long in large binary operations and the format conversion issues which will increase required clock cycles in the designs while converting the carry save format to binary representation are reviewed. Future scope of the design modification for reduced delay in critical path and hardware cost when compared with previous designs are discussed.**

Keywords—Montgomery modular multiplier, public key cryptosystem, carry save addition.

I. INTRODUCTION

The expansion in data communication and Internet services, like electronic commerce created a great demand of security in protecting sensitive data while in electronic data transmission. To ensure the required security, Public key cryptosystem [1], [2] are used by these systems. In the Public key cryptosystem, modular multiplication (MM) of large integers is a tedious and time consuming operation. Thus various algorithms and designs have been developed to carry out the MM more efficiently, out of which Montgomery's algorithm [3] is found to be the best where the trial division will be replaced by addition and shifting operations ie, the quotient is determined from least the significant bits of the operands and replaces the division by series of shifting modular addition. Thus these operations can be implemented in VLSI designs and a more efficient, fast MM outcome can be obtained.

The Montgomery's algorithm takes lengthy carry propagation for large binary operands which increases the execution time of multiplication. To overcome this problem, various designs based on the carry save addition are implemented to obtain fast results during the multiplication. In the approaches reviewed, it is found that the designs can be classified into two based on the use of inputs and outputs. The former make use of the inputs directly whereas the later inputs in the carry save format.

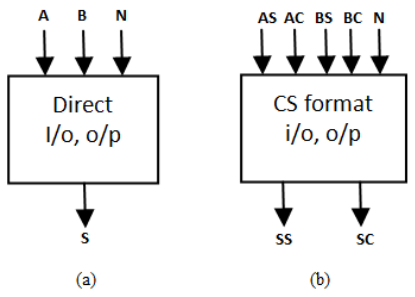

Fig. 1. (a)Direct inputs (outputs) (b) carry save format inputs (outputs)

In the direct input output (DIO) method [4]-[7] in Fig. 1.(a) the inputs (for ex. A, B, N) in binary form are taken as it is and output (S) operands are in same format, for the multiplication process. The intermediate results are in the carry save format so that carry propagation can be avoided, later the format is converted back into required binary representation by carry propagation adder (CPA) or by using different CSAs. Whereas in the case of carry save input output format (CSIO) [8], [9] in Fig. 1. (b), inputs are represented as AS, AC, BC, BC and the output as SS, SC so that the format conversion can be avoided thus the multiplication output can be obtained with a reduced clock cycles. But in this method the inputs increase, the registers assigned for the inputs increase, CSAs used increases thus the Montgomery Modular Multipliers based on this method have complex hardware and long propagation compared to the DIO multipliers. Different varieties of Modular MM are reviewed and future scope of improvement mainly by using one CSA to reduce the critical path, hardware complexity and enhance the performance by combining different CSA architectures [11]-[12] are discussed in this paper.

The paper is arranged with Section II explaining various Montgomery Modular multiplier architectures with their structures and algorithm. Section III discusses the results and analysis made. Conclusion is given in Section IV. Finally the enhancement that can be done in the present designs are mentioned briefly.

II. REVIEW OF MONTGOMERY MODULAR MULTIPLICATION ALGORITHMS

A. *Montgomery Modular Multiplication*

Most of the public key cryptosystems make use of RSA strategy in which the multiplication of A.B (mod N) is a crucial operation (let N > 0 be an odd integer). Achieving the reduction of A.B (mod N) is time consuming compared to multiplication of A.B without reduction. Montgomery introduced a new algorithm for calculating products (mod N) with quick output and low cost compared to other techniques of reduction(mod N),known was Montgomery Modular Multiplication. Montgomery multiplication of A and B (mod N), denoted by MM(A, B, N) defined as $A \times B \times R^{-1}$(modN), where $R = 2^n$ and n can be calculated as shown in the algorithm, $n = (\log_2 N) + 1$

Algorithm 1 Montgomery Modular Multiplication

```
Consider N(odd), $n = (\log_2 N) + 1$
Inputs: A, B, N(mod)
Output: S[n]
1. S[0]=0
2. for i=0 to n-1 {
3. $q = (S[i] + A_i \times B) \bmod 2$
4. $S[i + 1] = (S[i] + A_i \times B + q_i \times N)=2$
5. }
6. if $(S[n] \geq N) S[n] = S[n] - N$
7. return S[n]
```

As Montgomery MM is not normal multiplication, always there will be a conversion from the normal domain multiplication to the Montgomery domain. The conversion from and to the normal domain and the Montgomery domain is given by the relation A to A' with $A' = A \times 2^n$ (modN). The conversion can be done as shown in the table follows:

Normal Domain	↔	Montgomery Domain
A	↔	$A' = A \times 2^n$ (modN)
B	↔	$B' = B \times 2^n$ (modN)
A × B	↔	$(A \times B)' = A \times B \times 2^n$ (modN)

TABLE I. Conversion in Normal Domain and Montgomery Domain.

Apart from the conversion cost in the initial part, if many Montgomery multiplications are preceded by an inverse conversion say, for an RSA we obtain an advantage over ordinary multiplication. The verification of the Montgomery MM algorithm is: Define S[i] as

$$S[i] = \frac{1}{2j}\left(\sum_{j=0}^{i-1} A_j \times 2^j\right) \times B \bmod N$$

The convergence range of the inputs in the MM algorithm are $0 \leq A, B < N$ similarly the convergence range of the output S is $0 \leq S < 2N$. Thus the values exceeding the limits is termed as oversized, and if $S \geq N$ then the additional operation $S = S - N$ is done, so the values will be in the required limits and not oversized. The main issue is in handling the comparison and subtraction procedure which makes the design more slow and complex. In order to overcome this issue C. D. Walter [10] proposed a method in the paper 'Montgomery exponentiation needs no final subtractions', by which the iteration limits of 'i' is from 0 to n+2 from the previous n-1 and the value of R is assigned like $2^{(n+2)}$ modN.

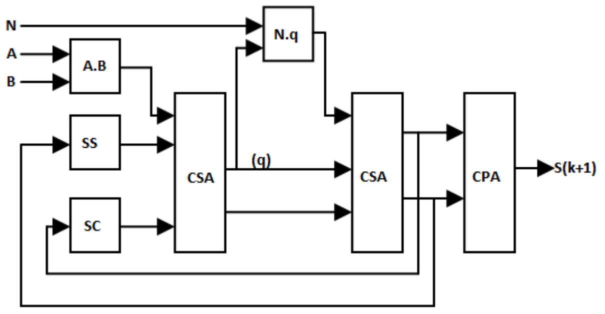

Fig. 2. 1024-BIT Montgomery MM

B. *Direct Input Output Based Montgomery Multiplication*

In the direct input output (DIO) based Montgomery MM the inputs and outputs in the same format are fed directly in the binary form without any change. The deigns which comes under this strategy are reviewed in a sequential order and are discussed in detail.

1) *Two CSA and a CPA:* First one discussed is "Implementation of 1024-bit Modular processor for RSA Cryptosystem" [4], in this architecture the main intention is in reducing the hardware complexity from the previous designs. The increment of the iteration process [10] discussed above in the algorithm is adopted in this architecture so as to eliminate the comparison and subtraction of the outputs that are over sized.

In the architecture, modular exponentiation is done by repeated Montgomery multiplication. In RSA cryptosystem working by Montgomery MM requires two additional processes to complete modular exponentiation. First is to convert input data A, B into (ABR) mod N from which the intermediate result of modular exponentiation is $(AB)^eR$ modN. After obtaining the intermediate result computation for the removal for R is done and the final result is obtained as $(AB)^e$ modN. In the process the limits because of the first bit in the exponent occurs which can be solved by setting the initial bit to 1 or by finding the position of the initial non-zero bit. This requires additional operation which the design will compute with increased number of modular multiplication. Since n is 1024 the small increase can be ignored?

The architecture as in Fig. 2. Consist of two CSA structures and a block to convert the format that is a 32-bit Carry Propagation Adder (CPA) along with a shift register. The adder gives a 32-bit output at all clock cycles, in the case of 1024 bit multiplier operation, 32 additional clocks cycles are required. In the architecture, the 1024 bit input register consists of 32-bit registers in which the values are shifted at all clock cycle and the inputs are refreshed. By using the CPA and shift register structure the cost of hardware is reduced and the shift register creates 32 bit in put output interface possible. But the CPA for format conversion enlarges the circuit which increases the hardware complexity and the propagation constant.

2) Two CSA without a CPA: In the subsection discussed earlier the architecture is having the CPA designs which make the multiplier suffer from the long critical path during the format conversion process. Whereas in "An efficient CSA architecture for Montgomery modular multiplication" by "Zhang, Zheng and Shao" [7] proposed a strategy that reuses a CSA to perform the format conversion done by the CPA thus reducing the critical path delay taken and remove the CPA entirely. The algorithm of this architecture is as discussed;

Algorithm 2 Two CSA without a CPA

```
Input: A, B, N with $0 \leq A, B < 2N$ and $2^{n-1} < N < 2^n$
Output: $Z = A \times B \times 2^{-(n+2)}$ modN
1. S = 0, C = 0
2. for i = 0 to n + 1 {
3. S, C = S + C + $A_i \times B$
4. S, C = S + C + $S_0 \times N$
5. S = S div 2; C = C div 2
6. While Carry! = 0 do
7. S, C = S + C
8. Return Z = S;
```

~~To perform the step 2 and step 7 in the algorithm two levels~~ of CSA are designed and one iteration will be executed during each clock cycles as shown in Fig. 3. The sum and carry output from the second CSA of an iteration is fed back into the first CSA for the next iteration.

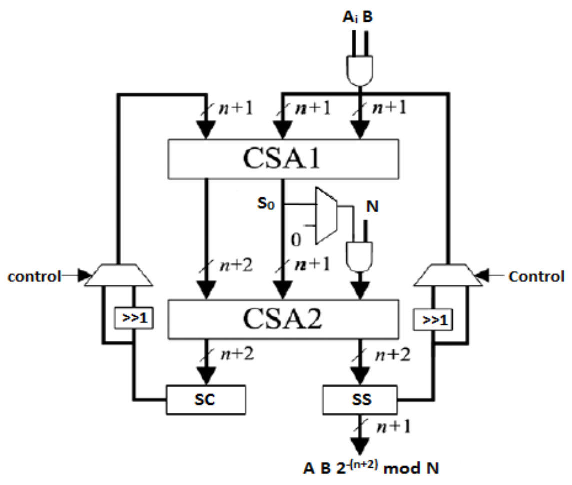

Fig. 3. Montgomery MM with only two CSA

In the first CSA the addition of sum, carry and product of A_i and B is carried out whereas in the second CSA the output from the first is summed with product of S_0 and N followed by division which is accomplished by shifting sum and carry by one bit. Then at the end of iteration both the sum and carry outputs of the second CSA are added to obtain the modular multiplication output without the use of any carry propagation logic but reusing the CSA. Because of the elimination of CPA, a fast operation is obtained compared to the previous architectures. But the clock cycles that are used extra for the conversion process depend on the longest propagation of the carry chain and in the worst case it can go up to n/2 clock cycles as two CSA designs are used.

C. *Carry Save format Input Output based Montgomery MM*

This is the second classification of Montgomery MM listed [8], [9] in this paper that is, the inputs are in carry save format thus the format conversion is avoided and the output will be calculated with fewer clock cycles. According to the strategy proposed by "McIvor, McLoone and McCanny" the values in the full process is kept in carry save format starting from the inputs to the outputs of the multiplier. The design mainly consist of three CSAs connected.

The architecture consists of three levels of carry save adders which are connected as shown in Fig. 4. The five inputs SC[i], SS[i], BS, BC, N are given to the CSAs respectively and input AC, AS is added in barrel shifter and outputs are given bit wise as shown in the figure. Final sum output is obtained at the sum register when carry becomes zero that is the sum of input vector bits. The outputs are obtained after each clock cycle by this approach. Based on the proposed architecture of multiplier the initial Montgomery MM algorithm is rewritten so as to obtain Algorithm 3 discussed below, Apart from the previous approach the operands are in CS format.

Algorithm 3 Five to two CSA

```
Input : AS, AC, BS, BC, N
Output : SS[K], SC[K]
1. SS[0], SC[0]
2. for i = 0 to n + 1{
3. $q_i = (SS[i]_0 + SC[i]_0) + (A_i \times (BS_0 + BC_0))\mathrm{mod}2$
4. $SS[i + 1], SC[i + 1] = (SS[i] + SC[i] + A_i \times (BS+$
   $BC) + q_i \times N)\mathrm{div}2$
5.}
6. Return SS[n], SC[n]
```

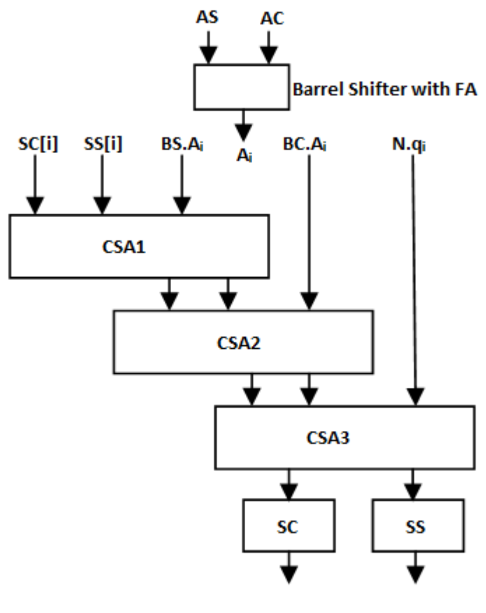

Fig. 4. CSIO Montgomery MM.

In the first algorithm, the inputs were A, B and the output S whereas now the operands are represented in carry save formats AS, AC, BS, BC, N and SS, SC respectively. While executing this algorithm the critical delay is found to be in the carry save addition. In algorithm 3 the inputs and outputs are in carry save format, and here A_i is to be determined. Therefore a full addition of AS and AC are performed to determine A_i which is achieved by using a full adder with barrel shifter with full adder. The Barrel shifter component will be explained in the next subsection.

Barrel Shifter component:

The full adder with barrel register finds the values of Ai required in the above two algorithms as shown in Fig. 5. In the Barrel shifter with full adder the Ai is determined by summing up the least significant bit of AS and AC operands by the full adder. The barrel shifter shifts the input bits concurrently to find the next value of Ai from the full adders thus keep the value ready for next iteration.

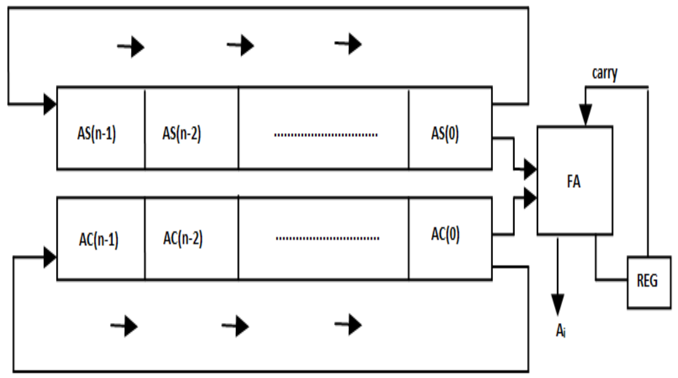

Fig. 5. Barrel Shifter with FA

The barrel shifter does not increase the critical delay in the algorithm as the operation is in parallel with the operation of CSAs and only one bit addition is required in each clock cycles. Thus there is no need of any additional hardware or clock cycles for the addition of AS and AC carry save format input operands to obtain Ai.

III. RESULT ANALYIS

The designs discussed in the previous section are modeled in Xilinx ISE using verilog and the simulation results are analyzed. First analysis is made based on the time taken by each designs. By giving some random value operands to the basic Montgomery modular multiplier the output obtained are kept as reference value. Here the operands A, B and N are assigned with 2, 3 and 8 respectively. When the same operand values are given to the “Two CSA and a CPA” design the output is obtained after eight clock cycles, for the design “Two CSA without a CPA” the output is obtained after five clock cycles and for the last design “Five to two CSA” the output is obtained after fourth clock cycle. Thus in the analysis based on the speed the Montgomery MM with CSIO gives faster results.

Second analysis is made on the hardware area required, for this the developed designs are analyzed in Cadence with 180 nm technology. The area analysis for each design is done, the result showed “Two CSA and a CPA” required 33800 sq. nm for the cell to occupy, “Two CSA without a CPA” design occupies an area of 14880 sq. nm and the design with three CSAs occupies 32657 sq. nm of area. The first design occupies the maximum area since it contains the complex CPAs within it, when we reuse the CSA to perform all the operation we obtain the least area. The area of the last design is also more compared to second design since there are three CSAs in the design.

IV. CONCLUSION

The CSIO multipliers helps in achieving the task by reusing the CSA by very few clock cycles and eliminating the format conversion process where all operands are in the CS format and the operation is faster compared to the DIO strategy. But the faster multiplication process is achieved by an increased hardware hence larger area compared to the DIO. In the DIO the number of clocks are more compared to CS format approach since they use CPAs for format conversion. To improve the performance of the DIO by increasing the iteration the comparison steps from the algorithm is eliminated and strategies like use of intermediate CS format, reusing the CSA technique were discussed. The analysis in the paper shows that the CSIO designs can be preferred if a fast design is required where reduction in area is not a constraint but if the area reduction is important in the design means the second design is preferable where the CSA is reused.

FUTURE SCOPES

The Montgomery Multiplier based on CSAs, the performance can be increased by maintaining a low complexity in the hardware. Instead of using more CSAs a single CSA can be used so as to do the summing in the iteration loop based on the algorithm, the pre computation of the values to be fed into the loop such as B+N operation [5], [6] and the long format conversion process by which the whole architecture will be having a small critical path and less hardware. The pre computation and the format conversion process may lead to additional clock cycles this can increase the critical path, so if the CSA can do a three input addition the additional clock cycles required for the mentioned processes can be made half. Thus the Montgomery multiplier will be having higher efficiency and the hardware takes small area.

REFERENCES

[1] Diffie W, and Hellman, “M. New directions in cryptography,” IEEE Trans. Inform.Theory IT-22, 644-654, (Nov. 1976).

[2] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signature and public-key cryptosystems”. Commun. ACM, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[3] P. L. Montgomery, “Modular multiplication without trial division,” Math. Comput., vol. 44, no. 170, pp. 519–521, Apr. 1985.

[4] 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.

[5] V. Bunimov, M. Schimmler, and B. Tolg, “A complexity-effective version of Montgomery’s algorihm,” in Proc. Workshop Complex. Effective Designs, May 2002.

[6] H. Zhengbing, R. M. Al Shboul, and V. P. Shirochin, “An efficient architecture of 1024-bits cryptoprocessor for RSA cryptosystem based on modified Montgomery’s algorithm,” in Proc. 4th IEEE Int. Workshop Intell. Data Acquisition Adv. Comput. Syst., Sep. 2007, pp. 643–646.

[7] Y.-Y. Zhang, Z. Li, L. Yang, and S.-W. Zhang, “An efficient CSA architecture for Montgomery modular multiplication,” Microprocessors Microsyst., vol. 31, no. 7, pp. 456–459, Nov. 2007.

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

[9] S.-R. Kuang, J.-P. Wang, K.-C. Chang, and H.-W. Hsu, “Energy-efficient high-throughput Montgomery modular multipliers for RSA cryptosystems,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 21, no. 11, pp. 1999–2009, Nov. 2013.

[10] C. D. Walter, “Montgomery exponentiation needs no final subtractions,” Electron. Lett., vol. 35, no. 21, pp. 1831–1832, Oct. 1999.

[11] A. Cilardo, A. Mazzeo, L. Romano, and G. P. Saggese, “Exploring the design-space for FPGA-based implementation of RSA,” Microprocess. Microsyst., vol. 28, no. 4, pp. 183–191, May 2004.

[12] D. Bayhan, S. B. Ors, and G. Saldamli, “Analyzing and comparing the Montgomery multiplication algorithms for their power consumption,” in Proc. Int. Conf. Comput. Eng. Syst., Nov. 2010, pp. 257–261.