2017 2nd IEEE International Conference On Recent Trends in Electronics Information & Communication Technology (RTEICT), May 19-20, 2017, India

# Design and Implementation of different architectures of Montgomery modular multiplication

Kavyashree S
M.Tech. VLSI Design & Embedded Systems, R.V.C.E.
Bengaluru-59
kavyasree131091@gmail.com

Dr. Uma B V
Prof & HOD, Dept. of ECE, R.V.C.E.
Bengaluru-59,
umabv@rvce.edu.in

**Abstract—The Montgomery multiplication is the main block of modular exponentiation in cryptography. This paper discusses the three different architectures of Montgomery Multiplication and their performance is compared in terms and area and time optimization. The architectures are designed to improvise the area and reduce time. The designs are implemented in Verilog HDL and simulated using Synopsys VCS. They are also synthesized in Synopsys Design Compiler using 45nm libraries to get the cell area. The experimental results show that the time required for one Montgomery multiplication measured in terms of number of clock cycles is reduced by 1.5%, 1% and 3.5 % for the three different architectures discussed here over the previous implementations. It is achieved by minimizing the signals controlling the critical path of the design. Also there is a reduction in total cell area due to technology for the three designs presented in this paper.**

***Keywords – CSA (Carry Save Addition), Montgomery Multiplication, Cryptography***

## I. INTRODUCTION

Cryptography is very essential for the security of the systems and network. Cryptographic algorithms involve many mathematical concepts, Modular Arithmetic being the most important and majorly used. Modular exponentiation is the critical step in widely used PKC (public key cryptography) algorithms like RSA and diffie Hellman. This is both time and resource critical step of the algorithms. Research in RSA algorithms reveal that more than 3/4th of the time is spent in modular exponentiation step [1]. Hence it demands for implementation of modular exponentiation using minimum hardware resources and also time required for this step to reduce cost and avoid the delay over the network. Modular exponentiation is built with modular multipliers. Hence the aim is to reduce time and resource requirement for one multiplication which in turn has its effect on exponentiation. Montgomery Reduction algorithm is widely used in the field of PKC as it replaces the complicated division to simple shifting operations and also because of its ease to implement in hardware [2]. There are many different implementations of modular multiplier using different modifications of Montgomery algorithm.

In this paper three different architectures are implemented and comparative study is presented in terms of area and speed. Section I is the introduction about the Montgomery multiplication. Section II gives a conceptual explanation of the Montgomery Algorithm. Section III explains the different architectures; Section IV gives the implementation details and simulation results. Section V presents result analysis and finally Section VI concludes with the conclusion.

## II. MONTGOMERY MULTIPLICATION

Montgomery Multiplication is an algorithm for performing faster modular multiplication which has major applications in cryptography. Montgomery multiplication is an algorithm for computing modular multiplication $a \times b \bmod n$ where a, b and n are positive integers [2]. Montgomery algorithm converts the numbers into a domain called Montgomery domain and performs the essential operations and converts back which removes need of division and also simplifies the multiplication operation.

The steps involved in the Montgomery Multiplication are:

1. Consider two integers a and b less than modulo n
2. Introduce a number R greater than n such that gcd (R, N) =1
3. Find two integers $R^{-1}$and $N^{-1}$ such that:
   $RR^{-1}-NN^{-1}=1$.
4. Transform the multipliers to Montgomery space by the following multiplication and reduction:
   $A=a \times R \bmod N$
   $B=b \times R \bmod N$.
5. Compute the Montgomery multiplication using the algorithm in Figure 1 which computes product of A and B giving the result S in Montgomery space that is $S=A \times B \times R^{-1} \bmod N$
6. The final result can be obtained by back transforming from Montgomery space:
   $s=S \times R^{-1} \bmod N$

978-1-5090-3704-9/17/$31.00 © 2017 IEEE

**Algorithm 1 Radix 2 Montgomery Multiplication**

***Inputs*** : $A, B, N$ *(modulus)*
***Output*** : $S[k]$

1. *Initialization* $S[0] = 0$;
2. *for* $i = 0$ *to* $k - 1$ {
3. $q_i = (S[i]_0 + A_i \times B_0) mod\ 2$;
4. $S[i+1] = (S[i] + A_i \times B + q_i \times N)/2$ ;
5. }
6. *if* $(S[k] \geq N)$ {
7. $S[k] = S[k] - N$;
8. }
9. *return* $S[k]$;

Figure 1: Montgomery Algorithm

## III. Montgomery Architectures

In the Algorithm 1 in Figure 1 step 4 and step 8 are the critical path as they are performed using CPA (carry Propagate Adder). It involves lengthy chain of carry propagation as the operands involved in cryptography are higher order bits. Walter proved that the final subtraction shown in step 7 of Figure 1 can be removed by increasing the value of R and number of iterations [4]. Step 4 can be performed using CSA which does not involve carry propagation [3]. The Montgomery Modular Multiplication (MMM) Algorithms are majorly developed base on CSA. Three algorithm are discussed here.

### *A. Intermediate Operand Carry Save MMM (IOCS MMM)*

IOCS architecture was introduced in [5].The features of IOCS MMM are:

1. Inputs of multipliers A ,B and modulus N are represented in the binary format in Montgomery domain
2. Intermediate operands are represented in Carry Save Format (CSF). This avoids the long carry propagation of the intermediate result S.
3. The final result is obtained by converting the CSF back to binary format using CPA.

Figure 2 shows the algorithm 2 IOCS MMM.

**Algorithm 2 IOCS based Montgomery Multiplication**

***Inputs*** : $A, B, N$ *(modulus)*
***Output*** : $S[k+2]$

1. *Initialization* $SS[0] = 0$, $SC[0] = 0$;
2. *for* $i = 0$ *to* $k + 1$ {
3. $q_i = (SS[i]_0 + SC[i]_0 + A_i \times B_0) mod\ 2$;
4. $(SS[i+1], SC[i+1]) = (SS[i] + SC[i] + A_i \times B + q_i \times N)/2$ ;
5. }
6. $S[k+2] = SS[k+2] + SC[k+2]$;
7. *return* $S[k+2]$;

Figure 2: IOCS MMM Algorithm

The disadvantage of this algorithm is that it uses CPA for the final conversion which increases the number of clock cycles required for one multiplication. For example for 1024 bit operands it requires 32 cycles to complete the conversion using the 32 bit CPA. Hence the time delay is more and delay increases for higher order bits. The circuit and implementation is discussed in the next section.

### *B. All Operand Carry Save MMM (AOCS MMM)*

AOCS MMM aims at reducing the time required for one multiplication. This was first developed by [6].
The features of AOCS are:

1. All the operands are represented in CSF and there are registers for storing them.
2. Hence there is no need of conversion after each stage and hence CPA is not required which reduces the number of clock cycles required to perform one multiplication.

The AOCS MMM algorithm is shown in Figure 3. Though this removed the conversion and increased the speed, it increased the area as it required more registers of higher bit length to save all the operands in CSF

**Algorithm 3 AOCS based Montgomery Multiplication**

***Inputs*** : $AS, AC, BS, BC, N$ *(modulus)*
***Output*** : $SS[k+2], SC[k+2]$,

1. *Initialization* $SS[0] = 0$, $SC[0] = 0$;
2. *for* $i = 0$ *to* $k + 1$ {
3. $q_i = (SS[i]_0 + SC[i]_0 + A_i \times (BS_0 + BC_0)) mod\ 2$;
4. $(SS[i+1], SC[i+1]) =$
   $(SS[i] + SC[i] + A_i \times (BS + BC) + q_i \times N)/2$ ;
5. }
6. *return* $SS[k+2], SC[k+2]$;

Figure 3: AOCS MMM Algorithm

### *C. IOCS_New MMM*

IOCS_New is an algorithm to achieve both area and time optimizations which uses the concepts of both IOCS and AOCS was developed in [7]. It is a more mathematically involved algorithm. The features are:

1. Critical path in algorithm 2 and 3 is a 4 operand addition in step 4. This requires 2 CSA. This is removed by using means to select the 4th operand using a multiplexer as shown is steps 8 through 11 of Figure 4.
2. A modified CSA (MCSA) is used which can perform a 3 operand CSA and a 2 operand serial addition. This reduces the number of clock cycles required to perform the multiplication.
3. Pre computation is involved to jump over few unnecessary iterations. This involves the transforming the operands A, B, N to $\hat{A}$,$\hat{B}$,$\hat{N}$ and calculating of $q_{i+1}$, $q_{i+2}$ and $jump_{i+1}$ using the equations shown in steps 13 through 15 of Figure 4.

This algorithm reduced the time required to compute one multiplication by a large amount and with a considerably less area. But this algorithm does not execute in fixed time as precomputation and jump depends on the operands.

**Algorithm 4 Modified Montgomery Multiplication**

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

1. $\hat{B} = B \ll 3$ ; $\hat{q} = 0$; $\hat{A} = 0$; $jump_{i+1} = 0$;
2. $(SS, SC) = (2O\_MCSA(\hat{B}, \hat{N}))$;
3. *while* $(SC != 0)$
4. $(SS, SC) = (2O\_MCSA(SS, SC))$;
5. $\hat{D}$=SS;
6. $i = -1$; $SS[-1] = 0$ ; $SC[-1] = 0$;
7. *while*( $i \leq k+4$) {
8. *if* $(\hat{A} = 0$ *and* $\hat{q} = 0)$ $x = 0$;
9. *if* $(\hat{A} = 0$ *and* $\hat{q} = 0)$ $x = \hat{N}$;
10. *if* $(\hat{A} = 0$ *and* $\hat{q} = 0)$ $x = \hat{B}$;
11. *if* $(\hat{A} = 0$ *and* $\hat{q} = 0)$ $x = \hat{D}$;
12. $(SS[i+1], SC[i+1]) = (3O\_MCSA(SS[i], SC[i], x))$;
13. $q_{i+1} = ((SS[i]_1 \oplus SC[i]_1) \oplus (SS[i]_0 \,\&\, SC[i]_0))$;
14. $q_{i+2} = (SS[i]_2 \oplus SC[i]_2 \oplus (q_i \& \widehat{N_2}) \oplus (SS[i]_1 \,\&\, SC[i]_1)$;
15. $jump_{i+1} = \sim(A_{i+1} \mid (SS[i]_1 \oplus SC[i]_1) \mid (SS[i]_0 \,\&\, SC[i]_0)$;
16. *if* $(jump_{i+1} = 1)$ {
17. $SS[i+2] = SS[i+1] \gg 1$;
18. $SC[i+2] = SC[i+1] \gg 1$;
19. $\hat{q} = q_{i+2}$;
20. $\hat{A} = A_{i+2}$;
21. $i = i + 2$;
22. }
23. *else* {
24. $\hat{q} = q_{i+1}$;
25. $\hat{A} = A_{i+1}$;
26. $i = i + 1$;
27. }
28. }
29. $\hat{q} = 0$;
30. $\hat{A} = 0$;
31. }
32. *while* $(SC[k+5] != 0)$
33. $(SS[k+5], SC[k+5]) = (2O\_MCSA(SS[k+5], SC[k+5]))$;
34. *return* $SS[k+5]$;

Figure 4: IOCS_New MMM Algorithm

## IV. Implementation

All the algorithms presented in the previous section are implemented using Verilog HDL and synthesized using Synopsys Design compiler. The implementation details along with simulation results are presented in this section. The correctness of the results is verified by implementing the modular multiplication using Java BigInteger class which is independent of any algorithms. The multiplier block diagram is shown in Figure 5 which is general over the three architectures. Once the operation is completed a signal done is raised which can be used as an interrupt when the multiplier is used with different systems. The multiplier consists of separate data path which implements the algorithm and control path which is an FSM and provide necessary control signals.

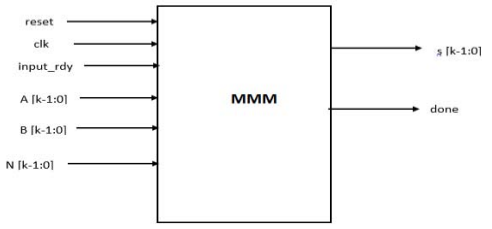


Figure 5: Block diagram of MMM

### A. IOCS MMM

The data path of IOCS MMM is shown in Figure 6. The architecture of IOCS MMM in [5] uses 2 multiplexers to compute step 3 and 4 of Algorithm 1 in Figure 1. Instead here both are replaced by and gates. This reduces the control signals required and hence the clock cycle is reduced. Also one shifter required to shift the sum and carry in step 4 can be implemented inside the CSA which further improves the clock reduction as this step is iterated several times in a loop. The control path is shown in Figure 7.

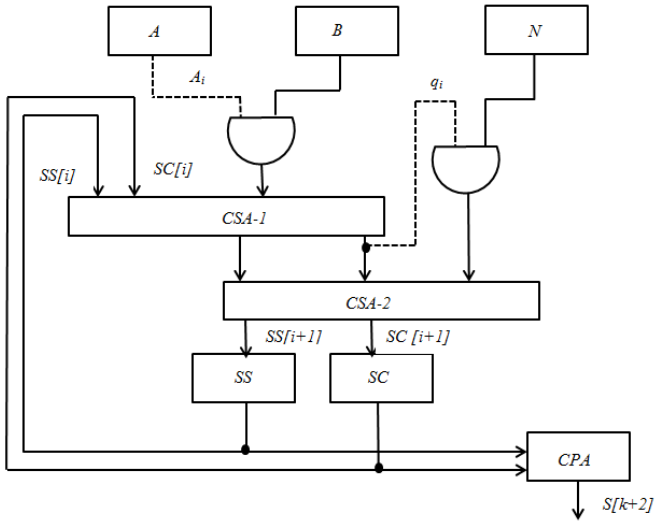


Figure 6: Data path of IOCS MMM

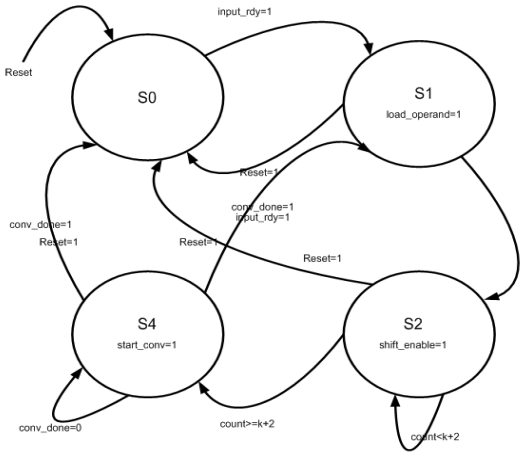


Figure 7: Control path of IOCS MMM

The simulation result for a 1024 bit multiplication is shown in Figure 8.

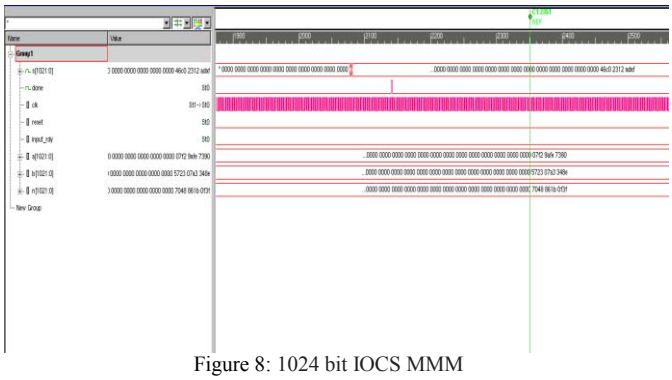

Figure 8: 1024 bit IOCS MMM

### B. AOCS MMM

AOCS is implemented as shown in Figure 9. The modifications applied for IOCS MMM are implemented for this also and an improvisation is obtained compared to [6].

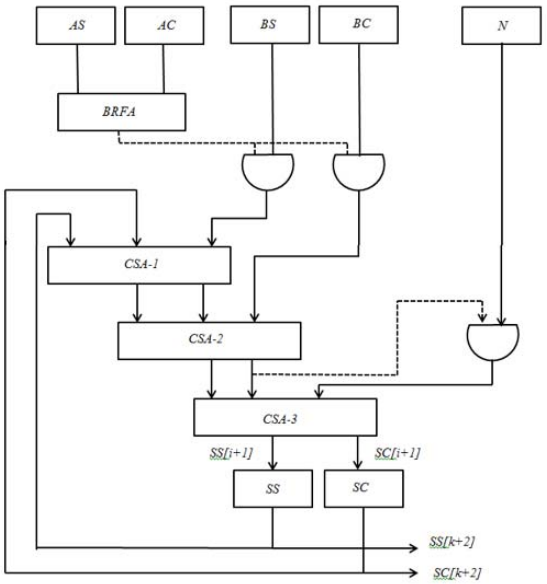


Figure 9: Data path of AOCS MMM

The control path is similar to the IOCS MMM explained in Figure 8 except that the last state of conversion does not exist.

The 1024 bit AOCS MMM is shown in Figure 10.

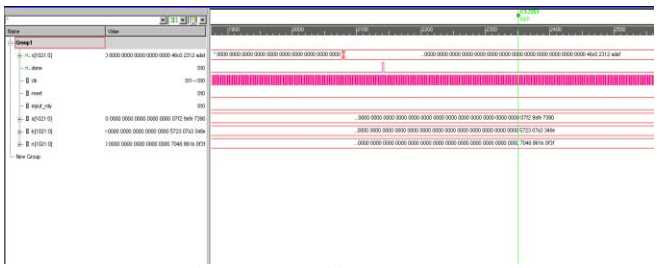

Figure 10: 1024 bit AOCS MMM

### C. IOCS_New MMM

This is a more improved architecture. The improvements performed over the earlier architectures can be carried to this also and the reduction over this will have a great affect and the speed is increased over [7]. Also the MCSA uses less multiplexers compared to the architecture of [7].The implementation is shown in Figure 11 and the corresponding simulation results are in Figure 12. But in this case the clock cycles consumed is not constant since the jump signal depends on the input data.

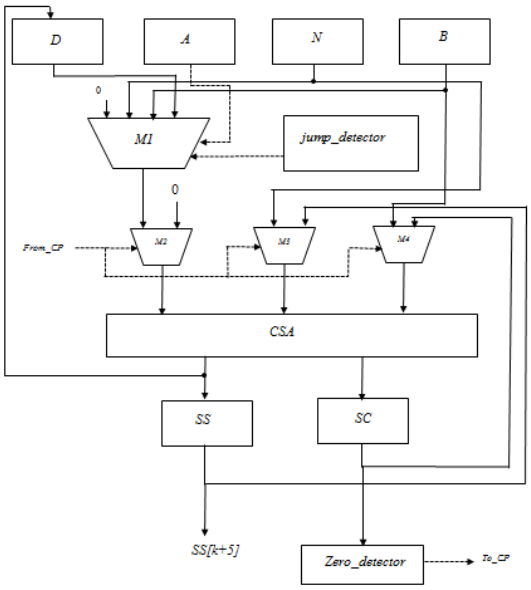


Figure 11: Data path of IOCS_NEW MMM

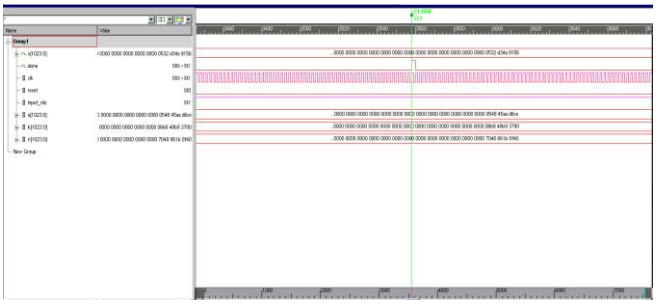

Figure 12: 1024 bit IOCS_NEW MMM

## V. RESULTS AND DISCUSSION

The time required for one multiplication is measured in terms of number of clock cycles. The number of clock cycles between the input_rdy when the multiplication starts and the done when the product is available gives the total time taken for MMM. The time taken is calculated in this way for the three design implementations shown in previous section. Table 1 shows the clock cycles required for the three architectures along with the earlier results from [7] for a 1024 bit multiplication. It can be seen that there is a reduction of clock cycles required for single multiplication which becomes more significant when this multiplier is used in exponentiation in

RSA algorithms. A chart showing the comparison is shown in Figure 13.

Table 1: Clock cycles consumed per multiplication

| Architecture | Clock Cycles(New) | Clock cycles[7] | Reduction (%) |
|---|---|---|---|
| IOCS | 1056 | 1072 | 1.5 |
| AOCS | 1019 | 1029 | 1 |
| IOCS_New | 850(average) | 880(average) | 3.5 |

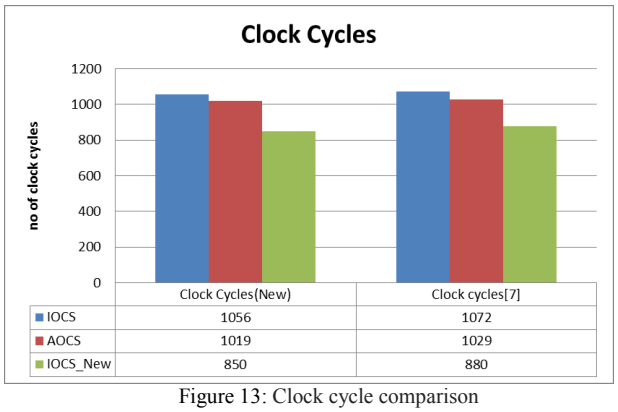


Figure 13: Clock cycle comparison

The designs are synthesized using Synopsys dc using 45nm libraries. The cell area are extracted and the corresponding cell area is shown in Figure 14. The chart shows that the relation discussed in the previous section holds good. That is area of AOCS is more than IOCS due to the use of more registers. The modifications and usage CSA reduces the area in case of IOCS_New.

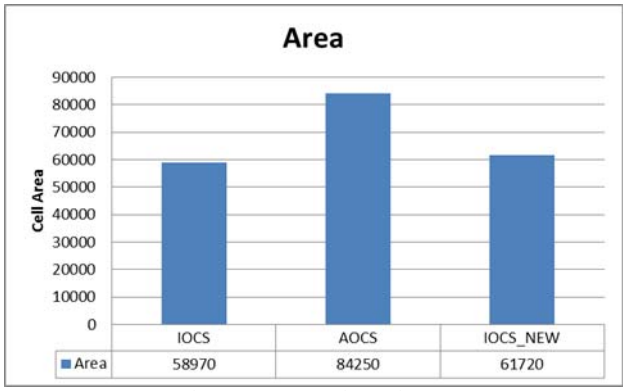


Figure 14: Comparison of Area

## VI. CONCLUSION AND FUTURE SCOPE

The results seems to best in case of IOCS_New as it is optimized both in area and time. But the disadvantage of this design is that that the time taken is not fixed and depends on the input operands. Hence the time taken becomes random for different applications. This is not suitable in applications requiring deterministic behavior. Hence the choice of the architecture depends on the application environment. Each of the three different architectures has advantages and disadvantages. IOCS can be used for area constrained applications where delay is tolerable, AOCS for extremely critical timing constraint. IOCS_NEW is used in cases where both time and area constraint are important and there is no requirement for deterministic behavior.

Area and speed can be further improved by using the concept of parallelism and pipelining. For the applications where area should be minimal, concept of pipelining can be used. It has to be carefully designed to reuse the same components for several operations. In these designs controls have to be properly generated from the control path. For applications where the area available is considerably large, speed up can be achieved by parallelism. In this several MMM can happen in parallel and hence the time required to complete the modular exponentiation reduces. Power dissipation should be carefully handled in these designs.

## REFERENCES

[1] Wangchen Dai, Donald Donglong Chen, Ray C. C. Cheung and Cetin K. Koc "Area-time Efficient Architecture of FFT-based Montgomery Multiplication," IEEE Transactions on Computers ,Year: 2016

[2] Peter L. "Modular Multiplication without Trial Division." Mathematics of Computation 4A, 170 (April 1985), 519-521

[3] Prof. Loh "Carry-Save Addition", Processor Design - Spring 2005 February 2, 2005

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

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

[7] Shiann-Rong Kuang, Member, IEEE, Kun-Yi Wu, and Ren-Yao Lu "Low-Cost High-Performance VLSI Architecture for Montgomery Modular Multiplication",IEEE Transactions On Very Large Scale Integration (VLSI) Systems, Vol. 24, No. 2, February 2016